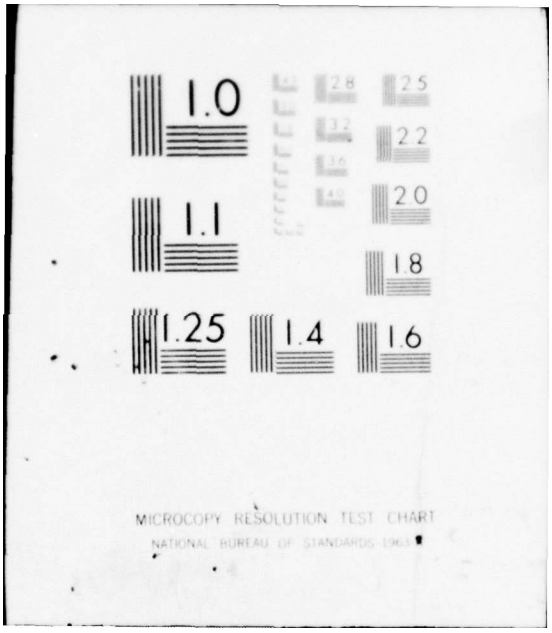
AD-A031 430 ILLINOIS UNIV AT URBANA-CHAMPAIGN COORDINATED SCIENCE LAB F/G 9/2
THE DESIGN OF TOTALLY SELF-CHECKING COMBINATIONAL CIRCUITS.(U)
AUG 76 J E SMITH DAAB07-72-C-0259
UNCLASSIFIED R-737 NL

1 OF 2
AD
A031430


1.0
1.1
1.25
1.4
1.6
2.8
2.5
3.2
2.2
3.6
2.0
4.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963

REPORT R-737 AUGUST, 1976 UILU-ENG 76-2225

CSL COORDINATED SCIENCE LABORATORY

THE DESIGN OF TOTALLY SELF-CHECKING COMBINATIONAL CIRCUITS

JAMES EDWARD SMITH

UNIVERSITY OF ILLINOIS - URBANA, ILLINOIS

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

(6) THE DESIGN OF TOTALLY SELF-CHECKING COMBINATIONAL CIRCUITS,

5. TYPE OF REPORT & PERIOD COVERED

Technical Report

6. PERFORMING ORG. REPORT NUMBER

(14) R-737; UILU-ENG-76-2225

7. AUTHOR(s)

(10) James Edward Smith

8. CONTRACT OR GRANT NUMBER(s)

(15) DAAB07-72-C-0259

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

(12) 146 P.

11. CONTROLLING OFFICE NAME AND ADDRESS

Joint Services Electronics Program

12. REPORT DATE

(11) August, 1976

13. NUMBER OF PAGES

138

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

(9) Doctoral thesis,

15. SECURITY CLASS. (of this report)

UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Self-Checking Circuits
Systematic Design Method

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The output reliability of logic circuits can be increased by constructing them such that they are "totally self-checking." Totally self-checking circuits have their outputs encoded in an error-detecting code. By considering the modeling of potential hardware failures, circuit structure and output codes, it is possible to construct circuits such that a failure within the circuit is indicated by a noncode output, and a code output indicates that the output is correct.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

097 700

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT (continued)

This thesis discusses the relationship between hardware faults, circuit structures, and output codings. The "path-fault secure" property is introduced and is used to develop a systematic design method for totally self-checking combinational circuits. The method places little restriction on the types of faults which may be assumed. Its applications to circuits with single and unidirectional fault assumptions are described in detail.

In addition, other concepts are discussed which deal primarily with the interconnection of small totally self-checking circuits to form larger ones. These concepts also lead to systematic methods for constructing totally self-checking check circuits.

76-2225
UILU-ENG ~~75-2224~~

THE DESIGN OF TOTALLY SELF-CHECKING COMBINATIONAL CIRCUITS

by

James Edward Smith

This work was supported in part by the Joint Services Electronics Program (U.S. Army, U.S. Navy and U.S. Air Force) under Contract DAAB-07-72-C-0259.

THE DESIGN OF TOTALLY SELF-CHECKING COMBINATIONAL CIRCUITS

BY

JAMES EDWARD SMITH

B.S., University of Illinois, 1972
M.S., University of Illinois, 1974

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1976

Thesis Advisor: Professor Gernot Metze

Urbana, Illinois

THE DESIGN OF TOTALLY SELF-CHECKING COMBINATIONAL CIRCUITS

James E. Smith, Ph.D.
Coordinated Science Laboratory and
Department of Computer Science
University of Illinois at Urbana-Champaign, 1976

The output reliability of logic circuits can be increased by constructing them such that they are "totally self-checking." Totally self-checking circuits have their outputs encoded in an error-detecting code. By considering the modeling of potential hardware failures, circuit structure and output codes, it is possible to construct circuits such that a failure within the circuit is indicated by a noncode output, and a code output indicates that the output is correct.

This thesis discusses the relationship between hardware faults, circuit structures, and output codings. The "path-fault secure" property is introduced and is used to develop a systematic design method for totally self-checking combinational circuits. The method places little restriction on the types of faults which may be assumed. Its applications to circuits with single and unidirectional fault assumptions are described in detail.

In addition, other concepts are discussed which deal primarily with the interconnection of small totally self-checking circuits to form larger ones. These concepts also lead to systematic methods for constructing totally self-checking check circuits.

ACKNOWLEDGMENT

I would like to thank Professor Gernot Metze, Mrs. Rose Harris, Mark Ketelsen, Trevor Mudge, David Ho, Jean Dussault, my wife Jerrie, my daughter Barbara, and my parents.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

1. INTRODUCTION

Several techniques for increasing the reliability of digital computers have been developed in recent years. Due to the large variety of applications and different cost considerations, no single technique is better than the others in all situations.

If a computer is to be used in an environment where it cannot be repaired, such as on board a space vehicle, massive masking redundancy such as quadded logic [1] and triple modular redundancy [2] or the more general n-tuple modular redundancy [3] are necessary. These methods require several times the hardware (at least three times) of the conventional simplex computer. In order to give some repair capability in this type of application, voted redundancy with standby spares [4] has been proposed. However, again, several times the hardware of a simplex machine is required, and the computer can only continue repairing itself for a finite period of time.

In ground-based applications where human intervention and repair are possible, the use of off-line testing [5] is prevalent. Here, a set of test cases is generated for a given computer. These tests are then run at intervals to determine if the computer has developed any failures since the tests were last applied. If a failure is detected, any work done by the computer since it was last tested is potentially in error and should probably be redone.

The advantage of off-line testing is that it requires no additional hardware--a very desirable attribute when hardware costs are high. The

obvious disadvantage is the time lost when the computer must be stopped for testing, and the time lost when computations must be redone after a failure has been detected.

In recent years the cost of hardware has dramatically decreased to the point that other test methods which require additional hardware, but which reduce lost computer time, deserve consideration. One such method is the use of computers which are self-checking [6]. That is, they constantly test themselves as they run. This self-checking capability is made possible by constructing the computer with circuits that have output information which is encoded. The outputs are then constantly monitored as the computer runs. A noncode output indicates that a failure has occurred.

Of course, in such a system, we would like to construct a circuit with an encoding such that all probable failures eventually result in a noncode output. Nevertheless, if a failure is indicated, it is possible that some output prior to the noncode output indicating the error was also erroneous (i.e. the wrong code output). For this reason it is desirable to design the self-checking circuit so that each probable failure not only yields a noncode output, but so that all outputs occurring before the non-code output are correct. Such circuits were defined by Anderson [7] to be "totally self-checking."

The additional circuit outputs due to the output coding and the monitoring, or "check", circuits require additional hardware. However, the only computing time which is lost is the time it takes to locate and repair a failure after the failure has been indicated. By carefully placing check circuits in key places, the time required to locate a failure can also be

reduced. Thus, the use of totally self-checking circuits is a good way of increasing reliability in an environment where it is important that system down time be as small as possible, and where hardware costs are relatively low. This is a situation which occurs very frequently in today's applications, for example in telephone switching and in industrial control.

As with most problems involving large systems, the problem of building totally self-checking digital systems can be simplified by decomposing a digital system into smaller building blocks. A major building block of a totally self-checking computer is the totally self-checking combinational circuit. This thesis will discuss the design of these circuits.

1.1. Literature Review

Most of the previous research in the area of totally self-checking combinational circuits has dealt with the design of circuits which perform a particular function. Possibly one of the first totally self-checking circuits was the adder in the IBM 650 as described in [8]. This adder, which used a biquinary code, appeared long before the term totally self-checking was formally defined by Anderson.

Elias [9] discussed the construction of self-checking circuits to perform bit-by-bit logical operations. Many researchers have considered totally self-checking adders using such codes as residue class codes [10], AN codes [11], parity codes [8], checksum codes [12], and two-rail codes [13].

In an early paper on totally self-checking design [6], Carter and Schneider discuss the construction of check circuits which are themselves totally self-checking (a necessity in a totally self-checking computer). In his Ph.D. thesis [7], D. A. Anderson gave designs for totally self-checking checkers for m-out-of-n codes. Later researchers [14,15] have done additional work on designing totally self-checking checkers.

As far as the design of arbitrary logic circuits other than adders or checkers is concerned, very little has been done. In fact the only "method" proposed thus far involved trial-and-error using fault simulation [16].

1.2. Thesis Summary

This thesis will deal with the design of totally self-checking combinational logic circuits. Specific design methods will be given for circuits which perform any arbitrary function. For this reason methods which work for only one particular function (such as adders) will not be discussed, although the general methods given here can be used to construct such circuits. Some emphasis will be given to the design of totally self-checking check circuits, since a totally self-checking circuit is of little value if it cannot be checked.

Chapter 2 describes the problem we are concerned with and presents the notation which will be used. The interconnection of functional blocks to form large totally self-checking circuits is discussed. The behavior and application of totally self-checking check circuits is

considered. Fault models are discussed, and the particular class of fault models with which we will be dealing is presented.

Chapter 3 considers the design of circuits which are totally self-checking with respect to unidirectional stuck-at faults. Codes and circuit structures which must be used under this fault assumption are given. Circuits and checkers are designed for several important codes.

Chapter 4 considers the design of circuits which are totally self-checking with respect to single stuck-at faults. It is shown how the circuits of Chapter 3 can be used in constructing circuits which are totally self-checking for this smaller set of assumed faults. Another class of circuits is discussed which use distance-two b-adjacent codes.

The circuits which are given in Chapters 3 and 4 provide some protection against faults other than those for which they were specifically designed. Chapter 5 discusses these other sets of faults. Included are shorts, transient, and intermittent failures.

2. BASIC CONCEPTS

2.1. Introduction

In this chapter, we will carefully define the problem we are attacking and will describe the notation used throughout the rest of this thesis. A few fundamental results on the interconnection and design of totally self-checking circuits will also be given. We will be considering multiple-input, multiple-output logic networks. These circuits will be composed of AND, OR, NAND, NOR, and NOT gates. They will contain no feedback, so they are all combinational. Although some circuits with feedback may be combinational [17], when we use the word combinational we intend it to exclude circuits with feedback.

The logic networks to be used will be denoted with the letter G. If more than one network is being discussed at a time, then we will subscript G to get networks G_1, G_2,... etc. If the network G has r inputs, then the input space of G is the set of all 2^r vertices of the r-cube. The input space of G is the domain of the logic function which G realizes. Similarly, if G has s outputs, then the set of all 2^s vertices of the s-cube is the output space of G and is the codomain of the function which G realizes. The range of the function realized by G may be (and often is) a proper subset of the output space.

The logic networks which will be considered will receive only a subset of their input space during "normal" (i.e. failure-free) operation. Thus, they are designed to realize a partial function from the 2^r inputs vertices onto the 2^s output vertices. The domain of this partial function

is the input code space and the range is the output code space. If the network G with input space X and output space Y realizes a partial function with input code space A, $A \subseteq X$, and output code space B, $B \subseteq Y$, then X-A is the input noncode space and Y-B is the output noncode space. We will often refer to a member of the input or output code space as a code word and to a member of the input or output noncode space as a noncode word.

Since logic networks are intended to realize a function, they will also be called functional blocks. Some functional blocks, such as adders, may appear to have more than one input code space or output code space; however, they may be concatenated into a single code space. Thus, all of the functional blocks considered will conceptually have a single input code space and a single output code space.

2.2. Modeling of Failures

We will study logic networks containing potential hardware failures which are of a permanent nature. These failures will be assumed to be of the type which can be modeled logically as "faults" consisting of sets of lines in the circuit which become stuck at logical values. This is the classical stuck-at fault model [18] where lines are stuck-at-1 (s-a-1) or stuck-at-0 (s-a-0).

Since each fault can be modeled as a set of stuck lines, a fault in a circuit which contains a set L of potentially faulty lines will be denoted as $\{\ell_1/d_1, \ell_2/d_2, \ldots, \ell_k/d_k\}$ where $\ell_i \in L$ and $d_i \in \{0,1\}$. If line ℓ_j is

s-a-0 (s-a-1) then $\ell_j/0$ ($\ell_j/1$) appears in the fault specification. We say that k is the multiplicity of the fault.

The set of faults which are most likely for a given logic network will be called its fault set. For the network G, this fault set will be called $\mathfrak{F}$, and for a network G_i, the fault set will be called $\mathfrak{F}_i$.

In the remainder of this chapter, we will discuss circuits with arbitrary stuck-at fault sets. However, in later chapters we will discuss two types of fault sets which can be described as follows: unidirectional fault sets where for any member of the set, L is the set of all gate input and output lines, all the d_i are equal, and any multiplicity is allowed; and single fault sets where for any member of the set, L is again the set of all gate input and output lines, and the multiplicity is one.

Practical experience tells us that more than one fault can be present in a logic network at the same time, and as more time elapses since the network was known to be fault free, the more faults we can expect in the network. Furthermore, the number of components in the network directly affects the number of faults which are present at a given time. Thus, logic circuits have a reliability which is a function of time and the number of logic components in the circuit. From this reliability, the mean time between failures (MTBF) [19] for a logic circuit can be computed.

This behavior will be modeled in a logic network G by assuming that the faults in $\mathfrak{F}$ occur one at a time with an interval roughly equal to an MTBF in between.

In some cases, the simultaneous existence of two members of $\mathfrak{F}$ in G might imply a contradiction. For example, one fault may contain $\ell_i/0$ and

and a second may contain $\ell_i/1$. To avoid such contradictions, we assume that after a fault involving some line occurs, no other fault is possible which involves the same line. To be more specific, we define the deletion of a fault f from a fault set $\mathfrak{F}$, $D(\mathfrak{F},f)$, to be a fault set such that $D(\mathfrak{F},f) = \{f' | f' \in \mathfrak{F}$ and no stuck line in f' is a stuck line in f$\}$. Then after a fault $f \in \mathfrak{F}$ has occurred in G, the fault set for G becomes $\mathfrak{F}' = D(\mathfrak{F},f)$.

Since all of our assumed faults permanently affect the logic function realized by the faulty network, a potentially faulty network G maps members of its input space X and members of its fault set $\mathfrak{F}$ onto members of its output space Y, i.e. $G: X \times \mathfrak{F} \rightarrow Y$. If network G maps input x to y under fault f, we will say $y = G(x,f)$. If no fault is present in G then the 'empty fault' will be denoted as ϕ, and G realizes $G(x,\phi)$.

All of the networks discussed in this thesis will be designed under the above assumptions. However, in Chapter 5, the effect of other failures on our networks not meeting the assumptions, e.g. nonpermanent or transient faults, will be discussed.

2.3. Basic Definitions

The ultimate goal of fault tolerant computing is to never get an erroneous output from a logic circuit. Due to the faulty behavior of hardware components, this goal cannot be attained. However, totally self-checking circuits make it possible to never give an erroneous output without also signaling the presence of a modeled fault which causes the erroneous output. It is important to notice the word "modeled" since the extent of

fault tolerance is only as good as the fault model; this will become clear later.

We now define exactly what is meant by "totally self-checking."

Definition 2.1: A functional block G with input code space A and output code space B is fault secure with respect to the fault set $\mathfrak{F}$ if

$$\forall f \in \mathfrak{F} \quad \forall a \in A \quad G(a,f) = G(a,\phi) \text{ or } G(a,f) \notin B.$$

That is, any erroneous output due to the presence of a member of $\mathfrak{F}$ must be in the output noncode space.

Definition 2.2: A functional block G with input code space A and output code space B is self-testing with respect to the fault set $\mathfrak{F}$ if

$$\forall f \in \mathfrak{F} \quad \exists a \in A \text{ such that } G(a,f) \notin B.$$

That is, each fault in $\mathfrak{F}$ causes some code input to be mapped to a noncode output.

Definition 2.3: A functional block G is totally self-checking (TSC) with respect to the fault set $\mathfrak{F}$ if it is fault secure and self-testing with respect to $\mathfrak{F}$.

Definition 2.3 is the conventional definition of the TSC property as originally given in [7]. However, there is a small amount of overlap between the fault secure and self-testing conditions. We remove this overlap by giving an alternative definition.

Definition 2.3': A functional block G is totally self-checking with respect to the fault set $\mathfrak{F}$ if it is fault secure with respect to $\mathfrak{F}$, and $\forall f \in \mathfrak{F} \quad \exists a \in A$ such that $G(a,f) \neq G(a,\phi)$.

The equivalence of these two definitions should be obvious since self-testing implies that $\exists a \in A$ such that $G(a,f) \neq G(a,\phi)$, and due to the fault secure property $G(a,f) \neq G(a,\phi)$ implies $G(a,f) \notin B$. Either of the definitions of TSC will be used depending on which is more convenient.

We see that if we have a circuit G which is TSC with respect to $\mathfrak{F}$, if only members of $\mathfrak{F}$ occur with a time equal to an MTBF between failures, and if all members of the input code space are applied within any time period less than an MTBF, then we have achieved our goal--the first erroneous output due to a failure is a noncode output which indicates the presence of the fault.

The preceding paragraph describes the ideal situation; if any of the listed conditions is not fully met, then we may fall short of the goal. The degree to which the conditions are met determines how close we come to the goal. If some failures occur which are not modeled by members of $\mathfrak{F}$, if two failures occur with much less than an MTBF between, or if the entire input code space is not cycled through often enough, an unnoticed erroneous output could easily result.

Thus, the fact that a circuit is TSC alone is not enough to guarantee the desired behavior. It is also necessary that all of the assumptions concerning the probability of a failure being in the fault set, the frequency of failures and the frequency of applying the input code space are met as closely as possible.

The circuits given in this thesis are only TSC with respect to specified fault sets. If all of the assumptions necessary are close to the actual conditions present in their application, they will behave satisfactorily.

There is a large class of circuits which are not all TSC, but for any given application would work equally well as far as achieving our goal.

Definition 2.4: A functional block G is effectively-totally self-checking (E-TSC) with respect to a fault set $\mathfrak{F}$ if

i) It is TSC with respect to $\mathfrak{F}$

or ii) it is fault secure with respect to $\mathfrak{F}$ and for any fault $f \in \mathfrak{F}$ which is not tested by input code words, the network G' (G with f present) is E-TSC with respect to $\mathfrak{F}' = D(\mathfrak{F}, f)$.

Circuits which are E-TSC but are not TSC have faults which may occur but which do not disturb the mapping of the input code space onto the output code space. Furthermore, any subsequent faults which may disturb the mapping never cause a code input to map to the incorrect code output, i.e. they only map to noncode outputs and indicate the presence of the fault.

It might appear useless to design circuits which are E-TSC but which are not TSC. However, we do not intentionally design such circuits -- sometimes when TSC networks are embedded in larger networks they fail to be TSC due to their surroundings but are still E-TSC. This phenomenon will be discussed in greater detail in the next chapter.

2.4. The Interconnection of TSC Functional Blocks

As pointed out by Anderson [7] it is often easier to build large TSC functional blocks from smaller ones than to design the large block as a unit. For this reason, we will not only be concerned with the design of

TSC functional blocks, but also with their interconnection. In this section some basic definitions and theorems concerning the interconnection of TSC functional blocks are presented.

When the interconnection of TSC functional blocks is considered, it is necessary to examine the mapping of noncode input words, since under failures in previous blocks a functional block might receive noncode inputs.

We define an error set E for a functional block G (E_i for a functional block G_i) to be a set of ordered pairs (p,q) where p is a member of the input code space of G and q is a member of the input noncode space. The significance of the ordered pair is that q represents an erroneous input which may be received by G instead of p. We will often give a word description of E such as "the set of all single errors."

Definition 2.5: A functional block G with output code space B is error secure w.r.t. the error set E if

$$\forall (p,q) \in E \quad G(q,\phi) \notin B \quad \text{or}$$
$$G(q,\phi) = G(p,\phi).$$

If G is error secure and receives an erroneous input contained in E then it either passes a noncode word on to subsequent blocks or corrects the noncode word to the code word which would have been produced if the previous blocks had all been fault free.

Definition 2.6: A functional block G with output code space B is error preserving w.r.t. an error set E if

$$\forall (p,q) \in E \quad G(q,\phi) \notin B.$$

A block which is error preserving is error secure, but the converse is not necessarily true.

Definition 2.7: A functional block is code disjoint if it is error preserving w.r.t. the error set E where

$$E = \{(p,q) | p \in A,\ q \notin A\}.$$

A code disjoint functional block maps all of the members of its input noncode space to noncode outputs.

The error preserving property is dependent entirely on a particular functional block. It is also interesting to consider what erroneous inputs a block might receive from its environment.

Definition 2.8: The potential error set $\mathcal{E}$ of a functional block G ($\mathcal{E}_i$ of a functional block G_i) is the set of pairs (p,q) where p is a member of the input code space of G, and q is a noncode input which can actually appear at G instead of p due to an assumed fault in previous blocks which feed G.

The previous blocks which feed G are all those blocks through which signals from the primary inputs must pass in order to reach G. Since only circuits with no feedback are being considered, there is no ambiguity concerning which blocks are previous to which. Thus, the potential error set of a functional block is entirely dependent on functional blocks which are previous to that block.

We now relate the set $\mathcal{E}$ and a set E for which a functional block G is error preserving.

Definition 2.9: A functional block G with potential error set $\mathcal{E}$ is error transmitting if it is error preserving w.r.t. E and $E \supseteq \mathcal{E}$.

It has been the practice [7] to specify that a functional block be code disjoint when it is merely required that the block be error transmitting. However, in cases where $\mathcal{E}$ is known (or a super set of $\mathcal{E}$ is known) it is possible to use blocks which are not code disjoint, but which are error preserving with respect to a sufficiently large error set. Of course, if nothing is known about the environment in which a block will be placed, the code disjoint property is required.

As we have said, all of our functional blocks have a single input space and a single output space. However, when blocks are connected to form a larger block, the input code space to some block may be the concatenation of the output code spaces of two or more previous blocks. When this occurs, there is a concatenation point at the input of the block being fed by two or more code spaces. If some output code space serves as a portion (or all) of the input code space of two or more blocks, there must be fanout at the output of the feeding block, and this is called a fanout point. It is also possible that some output code space may be divided into two or more independent parts with each part serving as at least a portion of the input code space to a subsequent block; where this occurs, there is a partition point. These ideas are illustrated in Example 2.1.

Example 2.1:

In Figure 2.1, a partition point is located at ①; fanout points are at ② and ④; concatenation points are at ③ and ⑤. Since the large block which we constructed is also considered to have one output code space, a concatenation point is also located at ⑥. □

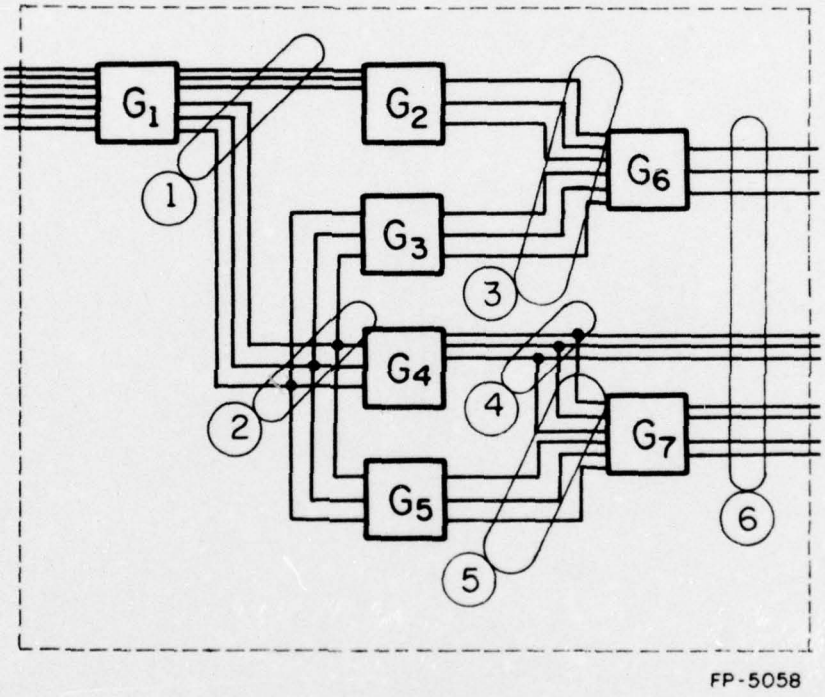


Figure 2.1. An interconnection of functional blocks.

In an interconnection of blocks, since all faults are at gate inputs or outputs, we can assume that all faults occur within the blocks and not in any of the interconnections. Thus, we see that at all times if a noncode word enters a fanout point, the same noncode word is propagated by each of the fanout branches. Similarly, if a noncode word enters any branch of a concatenation point, the output of the concatenation point must also be a noncode word. However, with partition points we do not get a similar result as is shown in Example 2.2.

Example 2.2:

At the partition point shown in Figure 2.2, let the code space which may enter at point A be

$$\{(0,1,0,1,0,1,1,0),(0,1,0,1,1,0,1,0),(0,1,1,0,0,1,0,1),(0,1,1,0,0,1,1,0),$$
$$(0,1,1,0,1,0,0,1),(0,1,1,0,1,0,1,0),(1,0,0,1,1,0,0,1),(1,0,0,1,1,0,1,0),$$
$$(1,0,1,0,0,1,0,1),(1,0,1,0,0,1,1,0),(1,0,1,0,1,0,0,1),(1,0,1,0,1,0,1,0)\}$$

This implies that the code spaces at points B and C both be

$$\{(0,1,0,1),(0,1,1,0),(1,0,0,1),(1,0,1,0)\}.$$

Now, if the noncode word (0,1,0,1,0,1,0,1) should enter at point A, the word which leaves point B is 0101 and the word which leaves point C is also 0101. Therefore, the error information has been lost at this partition point. □

Situations such as those shown in Example 2.2 demonstrate the necessity for the following definition when the interconnection of blocks is being considered.

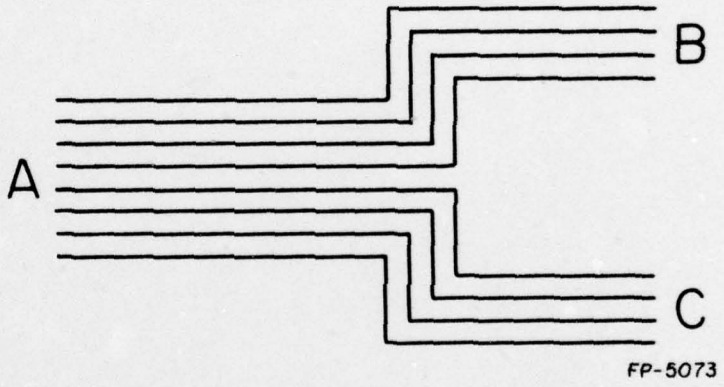


Figure 2.2. A partition point.

Definition 2.10: A partition point is error preserving if for any noncode input to the point at least one of the outputs is a noncode word.

In our earlier discussion it was assumed that the entire input code space of a functional block was applied at its input during normal (failure-free) operation. However, this assumption can be relaxed somewhat. This assumption was required to guarantee that the functional block was completely tested; however, for some particular realization of a functional block, a proper subset of the input code space may be sufficient to test for all failures. This brings us to our next definition.

Definition 2.11: A functional block with input code space A which is TSC for a prescribed fault set is sufficiently exercised if during normal operation a set of input code words $A' \subseteq A$ is applied to its input such that $\forall f \in \mathfrak{F} \quad \exists a \in A'$ such that $G(a,f) \neq G(a,\phi)$.

We now come to our first theorem concerning the interconnection of blocks to form large TSC blocks.

Theorem 2.1: An interconnection of functional blocks $G_1, G_2, \ldots, G_k$ with assumed fault sets $\mathfrak{F}_1, \mathfrak{F}_2, \ldots, \mathfrak{F}_k$ containing no feedback is TSC for the fault set $\mathfrak{F}_1 \cup \mathfrak{F}_2 \cup \ldots \cup \mathfrak{F}_k$ if

i) $\forall i$ G_i is TSC with respect to $\mathfrak{F}_i$.

ii) $\forall i$ G_i is sufficiently exercised.

iii) All the partition points are error preserving.

iv) All the G_i which are not fed from fanout points alone or primary inputs alone are error transmitting.

v) If no branch of a fanout point is part of the primary output, then at least one block fed by that fanout point is error transmitting.

Proof: We begin by showing that if a noncode word appears at the output of a functional block then it must propagate a noncode word to the primary outputs. Since we are proving the block is TSC for the union of the fault sets, it is apparent that any fault can only affect lines in one functional block. This, along with the fact that there is no feedback in the interconnection, implies that if a block receives a noncode input, some previous block is faulty and that the block receiving the noncode input and all subsequent blocks are fault-free.

If a noncode word enters a fanout point then it is obviously propagated to all of the fanout branches. If a fanout branch makes up part of the primary output, then the noncode word has been propagated to the output. Otherwise, due to condition v) at least one functional block which it feeds must propagate the error.

If a noncode word which is not on a fanout branch enters a functional block then due to condition iv) it is propagated as a noncode word to the output of the block.

If a noncode word enters a concatenation point, the output of the point is obviously a noncode word.

Finally, if a noncode word enters a partition point, due to condition iii) a noncode word is propagated to some functional block which the point feeds.

Inductively, we see that a noncode word must be propagated to the output since each component of the interconnection of blocks causes a noncode word to be propagated through it.

To show the connection of blocks is fault secure, without loss of generality assume block G_i is faulty. Then since G_i is the only faulty block in the network, and since it is fault secure, it either generates a noncode word for a given input which is then propagated to the output, or it generates the correct code output which can only result in a correct primary output. That the same is true for all members of $\mathfrak{F}_1 \cup \mathfrak{F}_2 \cup \ldots \cup \mathfrak{F}_k$ follows. Therefore, the connection is fault secure.

Since each block is self-testing and sufficiently tested, any member of $\mathfrak{F}_1 \cup \mathfrak{F}_2 \cup \ldots \cup \mathfrak{F}_k$ results in a noncode word at the output of the block in which it occurs under some input. Then due to the previous discussion a noncode word is propagated to the primary output. Thus, the interconnection of blocks forms a block which is self-testing for $\mathfrak{F}_1 \cup \mathfrak{F}_2 \cup \ldots \cup \mathfrak{F}_k$. Q.E.D.

Theorem 2.1 gives sufficient conditions for which an interconnection of blocks is TSC. However, there are interconnections of blocks which fail the above conditions but which are still TSC. To place some limit on these interconnections, we now give necessary conditions.

Theorem 2.2: An interconnection of functional blocks $G_1, G_2, \ldots, G_k$ with assumed fault sets $\mathfrak{F}_1, \mathfrak{F}_2, \ldots, \mathfrak{F}_k$ is TSC with respect to the fault set $\mathfrak{F}_1 \cup \mathfrak{F}_2 \cup \ldots \cup \mathfrak{F}_k$ only if

i) $\forall i$ G_i is sufficiently exercised.

ii) $\forall i$ G_i is self-testing.

Proof: If i) is not met then there must be some fault in $\mathfrak{F}_1 \cup \mathfrak{F}_2 \cup \ldots \cup \mathfrak{F}_k$ which is not tested during normal operation. Therefore, the connection of blocks cannot be self-testing.

If ii) is not met then, likewise, there must be some fault in $\mathfrak{F}_1 \cup \mathfrak{F}_2 \cup \ldots \cup \mathfrak{F}_k$ which is not tested during normal operation. Q.E.D.

The most interesting implication of Theorem 2.2 is what is not necessary. In particular it is not necessary that an arbitrary functional block in an arbitrary connection be fault secure. Consider a functional block G_2 fed by block G_1.

Now, say $G_1(a,\phi) = c$ and $G_1(a,f_1) = d$ where $a \in A$, the input code space of G_1; $f_1 \in \mathfrak{F}_1$; c and $d \in B$, the output code space of G_1; and $c \neq d$. Then G_1 is clearly not fault secure. However, if $G_2(c,\phi) = G_2(d,\phi)$ then the interconnection may be fault secure.

Even if a block does not feed another block, it does not necessarily have to be fault secure since it may make up part of the output word where there is dependency among the spaces which are concatenated to make up the output code space. In such a situation we may have something similar to a partition point which is not error preserving, only in reverse.

At this point we could give conditions under which the connection of functional blocks is E-TSC. However, without further limiting our fault model we cannot give any simple conditions which are more general than those given for TSC connections. In later chapters when specific fault sets are discussed, we will give such theorems for connections which are E-TSC for those particular fault sets.

2.5. TSC Check Circuits

Given a properly constructed TSC functional block, all of the diagnostic information which is needed can be acquired by examining the coded output of the block. This might require that a fairly large number of bits be examined. For this reason it is desirable to reduce the number of bits which need to be examined in order to determine if the functional block is behaving properly. This can be done by constructing a check circuit which takes the output of the functional block being checked as an input and yields an output which indicates the correctness or incorrectness of the word being checked. However, any such check circuit is presumably made of the same potentially faulty gates as the functional block. This problem can be overcome if the check circuit is totally self-checking itself.

This reasoning leads to the totally self-checking model as proposed by Anderson [7] and shown in Figure 2.3.

We now give a more formal definition of a TSC checker.

Definition 2.12: A totally self-checking checker for a code space A from a functional block G_1 is a functional block G_2 with an assumed fault set $\mathfrak{F}_2$ which takes A as its input code space and has an output code space which requires the fewest bits such that it is

i) TSC with respect to $\mathfrak{F}_2$ and

ii) error transmitting.

In other words, a TSC checker behaves in the following way. If the checker yields an output which is a member of its output code space then the output of the functional block being checked is correct. If an output

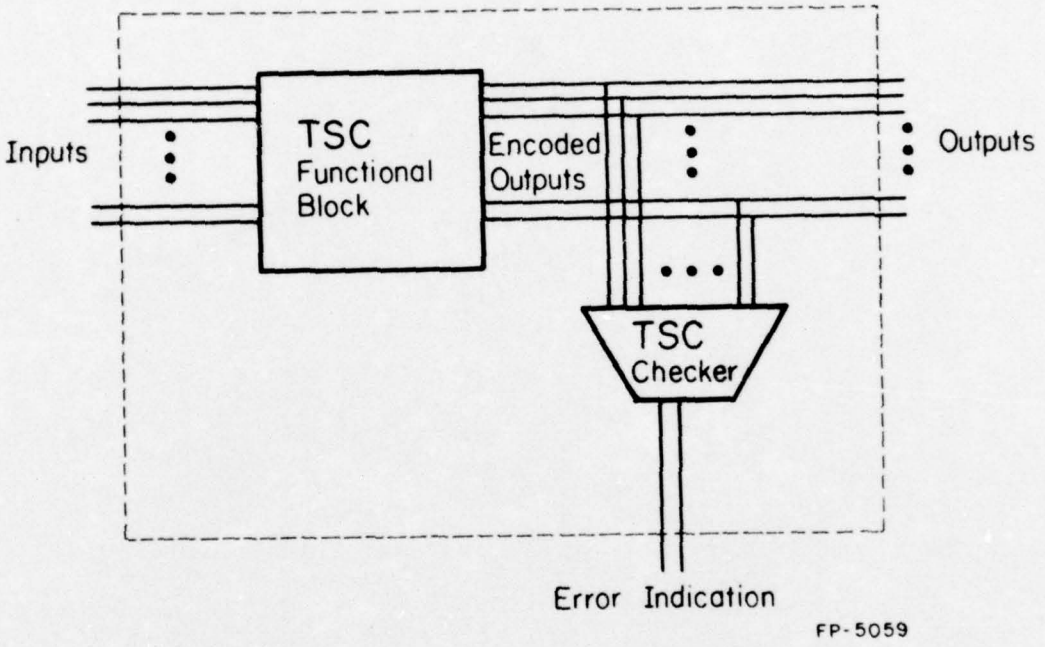


Figure 2.3. The totally self-checking model [7].

appears which is not a member of the checker's output code space then the functional block being checked has output an erroneous word and must be faulty, or the checker itself is faulty.

We observe that under our assumptions a checker used as indicated above does not need to be fault secure in order to produce the desired result. This is because we only are interested in whether or not the checker's output is a code or noncode word, not which code word it is. However, checkers have conventionally been TSC, and if we make them TSC we can follow the same design rules as for other TSC blocks. For this reason we will use checkers which are totally self-checking. Nevertheless it should be remembered that a self-testing checker would also be satisfactory for this application.

Definition 2.12 differs somewhat from the usual definition of a TSC checker. In [20] Diaz's definition allows any number of outputs as long as the functional block is code disjoint. We choose the more restrictive condition that it have a minimum number of outputs since its purpose is to reduce the number of bits which must be examined in order to extract the fault information from the block being checked. We use the slightly less restrictive condition that it be error transmitting, instead of code disjoint, since it only needs to check for errors which can potentially occur under the fault assumption. If it is not known a priori which errors can occur, we must assume any can occur, which makes it necessary to use the code disjoint condition. However, in many cases information about the errors which may occur is available and thus a less restrictive condition must be met.

The number of outputs required by a checker is obviously at least one. Fault sets which allow the construction of single output checkers (asymmetric stuck-at faults [21]) have been proposed. However, for current technologies it is doubtful that such a fault set models all likely hardware failures. Checkers for the fault sets to be discussed in the following chapters will require two or more outputs.

There are two applications of TSC checkers in TSC networks other than the one previously cited. In a large TSC network such as that shown in Figure 2.1 with a checker on the output, if a noncode output is received from the checker, we know that there is a failure somewhere in the network. Therefore, to locate the faulty functional block it might be necessary to test all of them. However, if checkers are placed throughout the network, e.g. one after each module, then the fault location problem becomes much simpler.

The other application occurs when we wish to interconnect functional blocks, but condition iv) of Theorem 2.1 is not met. This situation could easily happen if we have a set of blocks which were designed to be merely TSC with all noncode words specified as don't cares. In this case, a checker can be placed at the input to such a non-error transmitting block, and if we make the checker another functional block which contributes to the primary outputs then we have 'eliminated' a block which failed condition iv), since this block now is fed entirely by a fanout point. (The same fanout point feeds the checker which is error transmitting by definition.)

Thus, TSC checkers can be used to reduce the number of bits which need to be examined to acquire fault information, to improve the resolution with which faults can be located, and to cause a connection to be transformed from one which fails the conditions of Theorem 2.1 to one which meets them.

2.6. General Design Methods for TSC and E-TSC Networks

We now present a property which a circuit may have due to its structure and output encoding. The property is called the "path-fault secure" property. Testing a circuit for the path-fault secure property is a very simple process as compared with testing for the fault secure or self-testing properties which can be rather complex. Furthermore, it will be shown that a path-fault secure network is always E-TSC, and that a path-fault secure circuit is either TSC or can be easily modified so that it is TSC. It is also easy to describe classes of circuit structures and codes which must be path-fault secure. Two such classes of circuit structures and codes are considered in detail in later chapters.

We begin by defining the path-fault secure property. To do so we must use an important concept from the area of fault diagnosis known as "path sensitization" [40]. A path is sensitized from a fault site to an output by an input vector if the occurrence of the fault causes the output to change under that input. For example, in the circuit shown in Figure 2.4, a s-a-1 fault on line ℓ_1 is sensitized to the output of the circuit. However, a s-a-1 fault on ℓ_2 is not sensitized to the output.

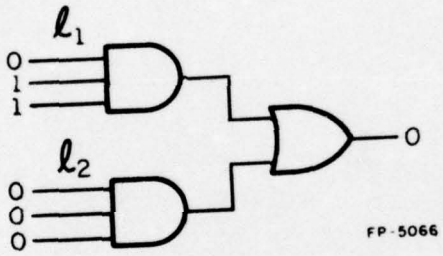


Figure 2.4

If there is an even number of inversions along a sensitized path from a fault site to an output and if a s-a-d fault causes a normal $\bar{d}$ on that line to go to d, the output also goes from a normal $\bar{d}$ to a d. If the number of inversions along a sensitized path is odd, a s-a-d fault causes a normal $\bar{d}$ on the line to go to d and causes the output to go from a normal d to a $\bar{d}$.

Let G be a network with outputs $y_1, y_2, \ldots, y_s$ and with fault set $\mathfrak{F}$ containing stuck faults. Then we define the sensitization set $\sigma(f)$, of a fault $f \in \mathfrak{F}$:

$\sigma(f) = \{(\alpha_1, \alpha_2, \ldots, \alpha_s) \mid \alpha_k \in \{0,1,n\}$;

$\alpha_k = 0 \Rightarrow$ some $\ell_j/0 \in f$ ($\ell_j/1 \in f$) such that there is some path from ℓ_j to output k with an even (odd) number of inverters;

$\alpha_k = 1 \Rightarrow$ some $\ell_j/0 \in f$ ($\ell_j/1 \in f$) such that there is some path from ℓ_j to output k with an odd (even) number of inverters, and if two component faults must have paths which have some line in common, there is no conflict as to the value which would be on that line if both faults were sensitized along the paths$\}$.

We call the vectors which belong to $\sigma(f)$ sensitization vectors.

The final restriction is required to take care of cases such as the one shown in Figure 2.5. Here, there is a path from ℓ_1 to y_1 with an odd number of inverters, and a path from ℓ_2 to y_2 with an even number of inverters; however, (1,0) is not in the sensitization set since a sensitized path from

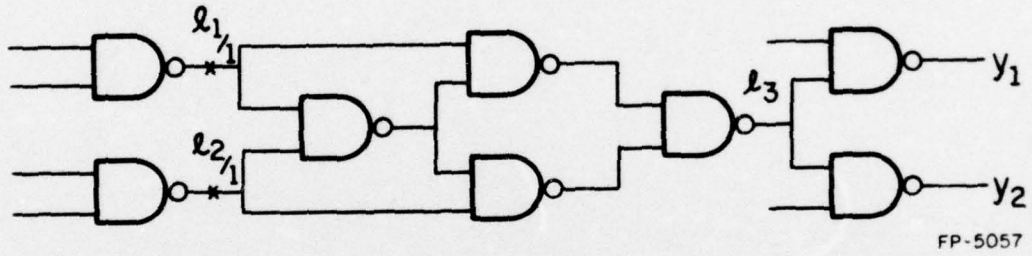


Figure 2.5

ℓ_1 to y_1 with an odd number of inversions requires that $\ell_3 = 0$, and a path from ℓ_2 to y_2 with an even number of inversions requires that $\ell_3 = 1$.

Example 2.3: In Figure 2.6 if $f = \{\ell_1/1\}$ then

$$\sigma(f) = \{(1,1,1,1),(1,1,1,n),(1,1,n,1),(1,1,n,n),(1,n,1,1),(1,n,1,n),(1,n,n,1),(1,n,n,n),(n,1,1,1),(n,1,1,n),(n,1,n,1),(n,1,n,n),(n,n,1,1),(n,n,1,n),(n,n,n,1),(n,n,n,n)\}.$$

If $f = \{\ell_2/1,\ \ell_3/0\}$ then

$$\sigma(f) = \{(n,1,1,n),(n,1,0,n),(n,n,0,n),(n,1,n,n),(n,n,1,n),(n,n,n,n)\}.$$

□

We observe that the sensitization vectors in $\sigma(f)$ indicate all the potential error patterns which can result from the fault f considering only the existence of paths which can propagate the effects of the fault to outputs. A 1(0) in some position of a vector indicates that it may be possible for the corresponding output to be forced to an erroneous 1(0).

In order to demonstrate the significance of sensitization vectors more explicitly, it is necessary to define a vector operator $\textcircled{e}$ which takes a member of the output space of G, say y, as its first argument and a sensitization vector as its second.

$$(y_1,y_2,\ldots,y_s)\textcircled{e}(\alpha_1,\alpha_2,\ldots,\alpha_s) = (\beta_1,\beta_2,\ldots,\beta_s)$$

where

$$\beta_i = y_i \quad \text{if} \quad \alpha_i = n$$
$$\beta_i = 0 \quad \text{if} \quad \alpha_i = 0$$
$$\beta_i = 1 \quad \text{if} \quad \alpha_i = 1.$$

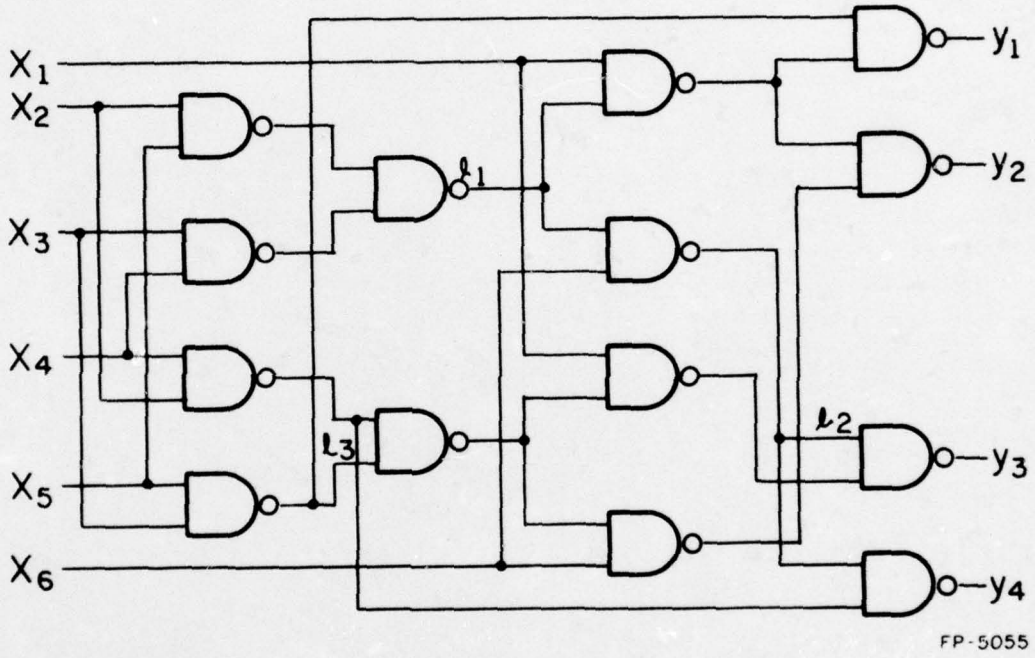


Figure 2.6. The network G in Example 2.3.

If y is a member of the output space and paths indicated by α are sensitized, then β is the resulting output.

In an obvious manner, we define

$$\{y \textcircled{e} \sigma(f)\} \text{ to be } \bigcup_{\alpha \in \sigma(f)} \{y \textcircled{e} \alpha\}.$$

Lemma 2.1: In a network G with input space X and fault set $\mathfrak{F}$

$$\forall x \in X \quad \forall f \in \mathfrak{F} \quad G(x,f) \in \{G(x,\phi) \textcircled{e} \sigma(f)\}.$$

Proof: If a set of paths from a fault f can be sensitized to a set of outputs under an input x then some sensitization vector in $\sigma(f)$ must indicate these paths, as we observed earlier. Therefore, the erroneous output vector must be a member of $\{G(x,\phi) \textcircled{e} \sigma(f)\}$. Q.E.D.

Example 2.4: If we apply some input x to the network G shown in Figure 2.6 such that $G(x,\phi) = (1,0,1,0)$ then if $f = \{\ell_2/1, \ell_3/0\}$ then $G(x,f) \in \{(1,0,1,0) \textcircled{e} \sigma(f)\}$ or $G(x,f) \in \{(1,1,1,0),(1,1,0,0),(1,0,0,0),(1,0,1,0)\}$. □

Definition 2.13: The sensitization set for a fault set, $\mathfrak{F}$, is $\Sigma(\mathfrak{F})$ where

$$\Sigma(\mathfrak{F}) = \bigcup_{f \in \mathfrak{F}} \sigma(f).$$

We say that $\{y \textcircled{e} \Sigma(\mathfrak{F})\} = \bigcup_{\alpha \in \Sigma(\mathfrak{F})} \{y \textcircled{e} \alpha\}$.

Definition 2.14: A circuit G with output space Y, output code space B, and fault set $\mathfrak{F}$ is path-fault secure (P-FS) with respect to $\mathfrak{F}$ if

$$\forall b \in B \quad \{b \textcircled{e} \Sigma(\mathfrak{F})\} \subseteq \{b\} \cup Y\text{-}B.$$

Lemma 2.2: A circuit G with output code space B and fault set $\mathfrak{F}$ is fault secure if it is path-fault secure.

Proof: From Lemma 2.1,

$$\forall a \in A \quad \forall f \in \mathfrak{F} \quad G(a,f) \in \{G(a,\phi) \textcircled{e} \sigma(f)\};$$

we know that $G(a,\phi) \in B$ and in particular, if we let $G(a,\phi) = b$ then

$$\forall a \in A \quad \forall f \in \mathfrak{F} \quad G(a,f) \in \{b \textcircled{e} \Sigma(\mathfrak{F})\}.$$

Since G is P-FS

$$\{b \textcircled{e} \Sigma(\mathfrak{F})\} \subseteq \{b\} \cup Y\text{-}B,$$

so

$$\forall a \in A \quad \forall f \in \mathfrak{F} \quad G(a,f) \in \{b\} \cup Y\text{-}B.$$

Therefore,

$$\forall a \in A \quad \forall f \in \mathfrak{F} \quad G(a,f) = b = G(a,\phi)$$

or

$$G(a,f) \in Y\text{-}B \not\in B. \qquad \text{Q.E.D.}$$

We will now show how the P-FS property relates to the E-TSC property.

Theorem 2.3: Any network G which is P-FS with respect to the fault set $\mathfrak{F}$ is E-TSC with respect to $\mathfrak{F}$.

Proof: If G has a fault $f_1 \in \mathfrak{F}$ such that $\forall a \in A \quad G(a,f_1) = G(a,\phi)$ then the circuit G' (G with f_1 present) maps members of A exactly as G did and has the fault set $\mathfrak{F}' = D(\mathfrak{F},f_1)$. If a second fault $f_2 \in \mathfrak{F}'$ is present and $\forall a \in A \quad G(a,f_1 \cup f_2) = G(a,f_1)$ then the circuit again maps members of A exactly as G did and has the fault set $D(\mathfrak{F},f_1 \cup f_2)$. This process can continue until, eventually, the circuit G" results with the fault set $\mathfrak{F}'' = D(\mathfrak{F},f_1 \cup f_2 \cup \ldots f_k)$ such that $\forall f \in \mathfrak{F}'' \quad \exists a \in A \quad G(a,f_1 \cup f_2 \cup f_k \cup f) \neq G(a,f_1 \cup f_2 \cup \ldots f_k)$.

We will now prove the theorem by induction on the value of k.

If k = 1 then $\forall f \in \mathfrak{F}$ $\exists a \in A$ such that $G(a,f) \neq G(a,\phi)$. Since G is P-FS it is also fault secure by Lemma 2.2, so Definition 2.3' is satisfied, and G is TSC and E-TSC by definition.

Now assume that if k = i and if G is P-FS then it is E-TSC. If k = i+1 the circuit G' (G with f_1 present) has a fault sequence k = i. Since G is P-FS, $\forall b \in B$ $(b \oplus \Sigma(\mathfrak{F}) \subseteq \{b\} \cup Y\text{-}B)$ by definition. G' has the fault set $\mathfrak{F}' = D(\mathfrak{F},f_1) \subseteq \mathfrak{F}$. Since f_1 can only reduce the paths in G, $\Sigma(\mathfrak{F}') \subseteq \Sigma(\mathfrak{F})$. Therefore $\forall b \in B$ $(b \oplus \Sigma(\mathfrak{F}') \subseteq \{b\} \cup Y\text{-}B)$. Thus G' is P-FS with respect to $\mathfrak{F}'$ and satisfies the induction hypothesis so G' is E-TSC. Therefore, from the definition of E-TSC, G must also be E-TSC. Q.E.D.

Now we will turn to TSC networks and consider how the P-FS property can be used for their construction. First, a few definitions are in order.

If a stuck-at fault occurs in a logic circuit, a simpler logic circuit which performs the same function as the faulty original can be formed by removing lines (and possibly gates) from the original. For example, if an input to a q-input AND gate is s-a-1, we can remove the line to get a (q-1)-input gate. This process of simplifying a logic network under a fault by removing lines (and gates) will be called the "reduction" of the network with respect to the fault.

In order to make it clear exactly which lines and gates may be removed, and to perhaps suggest an algorithm for doing so,we now define what we mean by reduction in a much more rigorous way. Some theorems pertaining to the reduction of circuits are also presented.

Definition 2.15: The expansion of the fault f within the network G, E(G,f), is a fault f' which can be constructed by the following steps. (A NOT gate is considered to be a single-input NAND or NOR gate.)

1) $f' \leftarrow f$.

2) Repeat steps 3-7 until none of them can be used to increase the size of f'.

3) If $\ell_i/0 \in f'$, ℓ_i is the input of an AND (NAND) gate, ℓ_j is the output of the AND (NAND) gate, and $\ell_j/d \notin f'$ for $d \in \{0,1\}$, then $f' \leftarrow f' \cup \{\ell_j/0\ (\ell_j/1)\}$.

4) If $\ell_i/1 \in f'$, ℓ_i is the input of an OR (NOR) gate, ℓ_j is the output of the OR (NOR) gate, and $\ell_j/d \notin f'$ then $f' \leftarrow f' \cup \{\ell_j/1\ (\ell_j/0)\}$.

5) If all the inputs to an AND (NAND) gate s-a-1 are in f', ℓ_j is the output of the AND (NAND) gate, and $\ell_j/d \notin f'$, then $f' \leftarrow f' \cup \{\ell_j/1\ (\ell_j/0)\}$.

6) If all the inputs to an OR (NOR) gate s-a-0 are in f', ℓ_j is the output of the OR (NOR) gate, and $\ell_j/d \notin f'$, then $f' \leftarrow f' \cup \{\ell_j/0,\ (\ell_j/1)\}$.

7) If $\ell_i/0$ ($\ell_i/1$) is the input line to a fanout point, ℓ_j is an output line of the fanout point, and $\ell_j/d \notin f'$ then $f' \leftarrow f' \cup \{\ell_j/0\ (\ell_j/1)\}$.

Thus, the expansion of a fault consists of determining additional lines in the network which are forced to take on constant values due to stuck lines which precede them in the network. We observe that the method for expanding a fault in a network must eventually terminate. This is because each of steps 3-7 adds a stuck line to f', once a stuck line is

added to f' it is never removed from f', and there is a finite number of lines in G which can become stuck. We should also point out that the final f' is the same regardless of the order with which faults are added to f' by steps 3-7; however, this fact is not necessary for the remainder of this section and will therefore not be proved.

Lemma 2.3: Let $f' = E(G,f)$ for any $f \in \mathfrak{F}$, then $\forall x \in X \quad G(x,f) = G(x,f')$.

Proof: The algorithm to generate f' begins with the fault $f' = f$ and adds elements to f' in a sequential manner in steps 3-7. It is obvious that $G(x,f) = G(x,f')$ before any of the steps 3-7 are applied. Let ℓ_j/d be the element added to f' at the kth application of any of steps 3-7. Then it is necessary to show that $G(x,f') = G(x,f' \cup \{\ell_j/d\})$.

Assume that $\ell_j/0$ is added to f' due to an application of step 3. This means there is an AND gate which has ℓ_j as its output and an input $\ell_i/0$. When input ℓ_i alone is s-a-0 the output of the AND gate is a constant 0. The same is true if ℓ_j is also s-a-0. Since all the other lines in G are unaffected, addition of $\ell_j/0$ cannot possibly alter the output of G under any input. Therefore, $G(x,f') = G(x,f' \cup \{\ell_j/0\})$. By using similar arguments, we can show that whatever element ℓ_j/d is added to f' due to an application of steps 3-7, $G(x,f') = G(x,f' \cup \{\ell_j/d\})$.

Thus since $G(x,f) = G(x,f')$ initially, and for each application of any of steps 3-7 this equality continues to hold, it must also hold when the algorithm terminates. Q.E.D.

Definition 2.16: If L is the set of lines in the network G, then $T(G,f) = \{\ell \mid \ell \in L$ and every path from ℓ to any primary output includes a line ℓ' such that $\ell'/d \in E(G,f)\}$.

We observe in the above definition that ℓ' may equal ℓ in which case ℓ must be in $T(G,f)$.

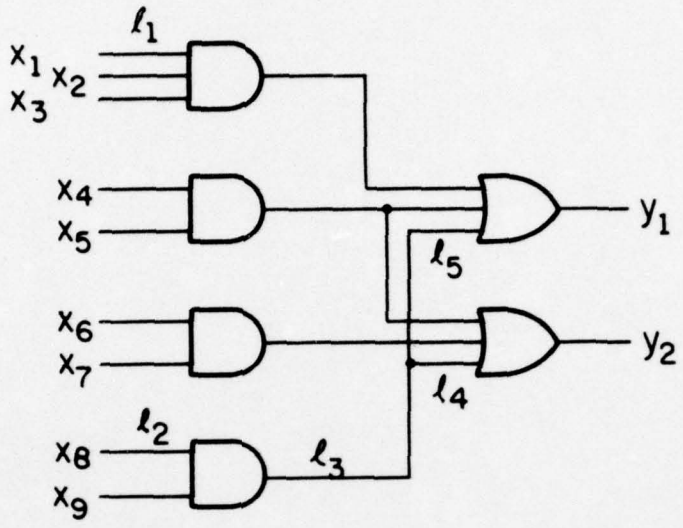
Definition 2.17: The reduction of a network with respect to a fault f, $R(G,f)$, is a network which contains all the gates in G whose outputs are not in $T(G,f)$, and the lines of G which are not in $T(G,f)$.

At this point, we should make some observations pertaining to Definition 2.17. First, if the reduction of a network forces the removal of primary input lines, then when a vertex is applied to the reduced network those coordinates which correspond to deleted inputs are simply ignored. Second, if the reduction of a network forces the removal of a primary output, then the output is assumed to take on the constant value at which it was stuck in $E(G,f)$ (it must have been a stuck line from Definition 2.16) and is assumed to be immune to any further failures. In the circuits which we will be considering, the removal of a primary output line can only happen in pathological cases. Finally, the reduction of a circuit may lead to such things as a single-input AND gate or OR gate or two consecutive inverters on a line. Such occurrences will not affect our later theorems; however, these structures can obviously be simplified without affecting our results. We retain them in order to avoid renaming lines which would also force the renaming of faults in the fault set for the circuit.

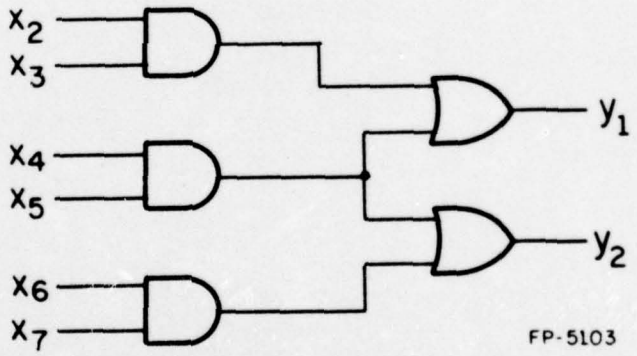
Theorem 2.4: If $G' = R(G,f)$ then $\forall x \in X \quad G'(x,\phi) = G(x,f)$.

Proof: From Lemma 2.3 $G(x,f) = G(x,E(G,f))$. Therefore, it is only necessary to show that $G'(x,\phi) = G(x,E(G,f))$. Lines and gates have been removed from G to get G' in one of two situations. First, excluding the line or gate output itself, all paths from the line or gate output to any primary output may contain a stuck line in E(G,f). In this case, the removal of the line or gate cannot possibly affect the primary outputs so the function must be unchanged by their removal. Second, the line or gate output itself may have been stuck and there is a path to a primary output from the line a gate output with no stuck lines in E(G,f). If the line or gate output is a primary output, the value of the primary output must be constant under all inputs, and we have said that removal of a primary output resulted in the output being permanently fixed at that constant value. Thus, the removal of a primary output line or gate must leave the function unchanged. Otherwise, the line or gate output must be a gate input since it cannot be a fanout point from the definition of E(G,f) and the fact that there is a path from the line to a primary output with no stuck lines in E(G,f). Furthermore, for the same reason, the gate input must be a s-a-1 input to an AND or NAND gate or a s-a-0 input to an OR or NOR gate, and the gate it feeds must have some other input line which is not stuck. In all of these cases the removal of the stuck line and the gate forming it cannot affect the function. Therefore $G'(x,\phi) = G(x,E(G,f))$. Q.E.D.

Example 2.5: Figure 2.7 shows the reduction of a network G with respect to the fault $\{\ell_1/1, \ell_2/0\}$. Figure 2.7(a) shows G with f present. E(G,f) is $\{\ell_1/1, \ell_2/0, \ell_3/0, \ell_4/0, \ell_5/0\}$. Figure 2.7(b) shows the reduced network.



(a)



(b)

Figure 2.7. The reduction of a circuit with respect to a fault.

After a circuit has been reduced, a number of faults in the fault set of the original circuit are no longer possible since they contain lines which are no longer present. For this reason, we must perform an operation on the fault set similar to the reduction performed on the circuit.

Definition 2.18: The simplification of a fault set $\mathfrak{F}$ with respect to a fault f is a fault set $S(\mathfrak{F},f) = \{f \mid f \in \mathfrak{F}$ and if $\ell_i/d \in f$ then $\ell_i \notin T(G,f)\}$.

Theorem 2.5: If G is P-FS with respect to $\mathfrak{F}$ and $\exists f \in \mathfrak{F}$ such that $\forall a \in A \quad G(a,f) = G(a,\phi)$ then the circuit $G' = R(G,f)$ is P-FS with respect to $\mathfrak{F}' = S(\mathfrak{F},f)$.

Proof: Reduction of a network can only decrease the number of paths and cannot change the inversion parity along any given path. Therefore, $\Sigma(\mathfrak{F}') \subseteq \Sigma(\mathfrak{F})$. Since G and G' have the same output encoding, $\forall b \in B \ (b \oplus \Sigma(\mathfrak{F}') \subseteq \Sigma(\mathfrak{F}) \subseteq \{b\} \cup Y\text{-}B)$. Then G' must be P-FS. Q.E.D.

The proof of Theorem 2.5 can be used to derive a procedure for constructing TSC circuits:

1) Construct a circuit G which is P-FS with respect to the required fault set $\mathfrak{F}$.
2) As long as the fault set has a fault f such that $\forall a \in A \quad G(a,f) = G(a,\phi)$, reduce G with respect to f and simplify $\mathfrak{F}$ with respect to f.

The final circuit must be fault secure with respect to the simplified $\mathfrak{F}$ according to Theorem 2.5 and Lemma 2.2. Furthermore, in order for the method to terminate, it must be true that $\forall f \in \mathfrak{F} \ \exists a \in A \quad G(a,f) \neq G(a,\phi)$. Therefore, by Definition 2.3 the circuit is TSC with respect to the simplified $\mathfrak{F}$. In addition, the reduced G performs the same mapping of input code words onto output code words as the original G did. This is

true because of Theorem 2.4 and the condition which must hold before a reduction with respect to a fault can be made.

The method given above will be used as the primary means of constructing TSC circuits in later chapters.

3. TSC CIRCUITS FOR UNIDIRECTIONAL FAULTS

3.1. Introduction

We now turn our attention to the first of two actual fault sets for which TSC circuits will be designed--unidirectional faults. A unidirectional fault is a set of any number of gate input or output lines which are all stuck at the same value.

We choose to study this type of fault because failures which can be modeled in this way are prevalent in some logical devices (e.g. memories). Also, because single faults, which will be covered in the next chapter, are a subset of unidirectional faults, the circuits which are given here are also TSC with respect to single faults. By using unidirectional faults when discussing these circuits, we can simply be more general.

Although fault locations are restricted to gate inputs and outputs, the circuits designed here will also be TSC for primary input faults since any unidirectional primary input fault can be modeled as a unidirectional gate input or output fault.

3.2. Basic Definitions and Theorems

As mentioned earlier, we consider an r-input, s-output logic network to have the 2^r vertices of the r-cube as possible input combinations, and the 2^s vertices of the s-cube as possible output combinations. A particular vertex of the r-cube is written as

$$a = (a_1, a_2, \ldots, a_r) \text{ where } a_i \in \{0,1\}.$$

A vertex with exactly k ones will be called a k-vertex. The complement of a vertex is formed by complementing each of its components.

The usual partial ordering on the vertices is defined as

$$a \leq b \quad \text{iff} \quad a_i \leq b_i \quad \text{for all } i.$$

For example,
$$(1,1,0,1) \leq (1,1,1,1)$$
$$(1,0,0,1) \leq (1,1,0,1)$$
$$(1,0,0,1) \not\leq (0,1,0,1).$$

We will say that a <u>covers</u> b (or b is covered by a) if $b \leq a$.

We see that the vertices of the r-cube and the partial ordering we have defined form a Boolean lattice [22]. A graphical representation of a Boolean lattice which we will use consists of the vertices of the r-cube as vertices of the graph with an edge between the vertices a and b if and only if $a \leq b$ and there exists no c such that $a \leq c \leq b$. We say that vertex a is <u>adjacent</u> to the vertex b if there is an edge connecting them in the lattice.

<u>Example 3.1</u>: The Boolean lattice formed by members of the 4-cube and the covering relation is shown in Figure 3.1. □

An <u>antichain</u> in a lattice is a set of distinct vertices A such that $\forall a \in A \quad \forall b \in A$ if $a \neq b$ then $a \not\leq b$. A <u>maximal antichain</u> is an antichain such that no vertex not in the antichain can be added to make a larger antichain. For example, $\{(1,1,1,0),(1,1,0,1),(1,0,1,1),(0,1,1,1)\}$ is a maximal antichain in the lattice of Figure 3.1. In later discussions the relation $<$ will be used whenever $a \leq b$ and $a \neq b$.

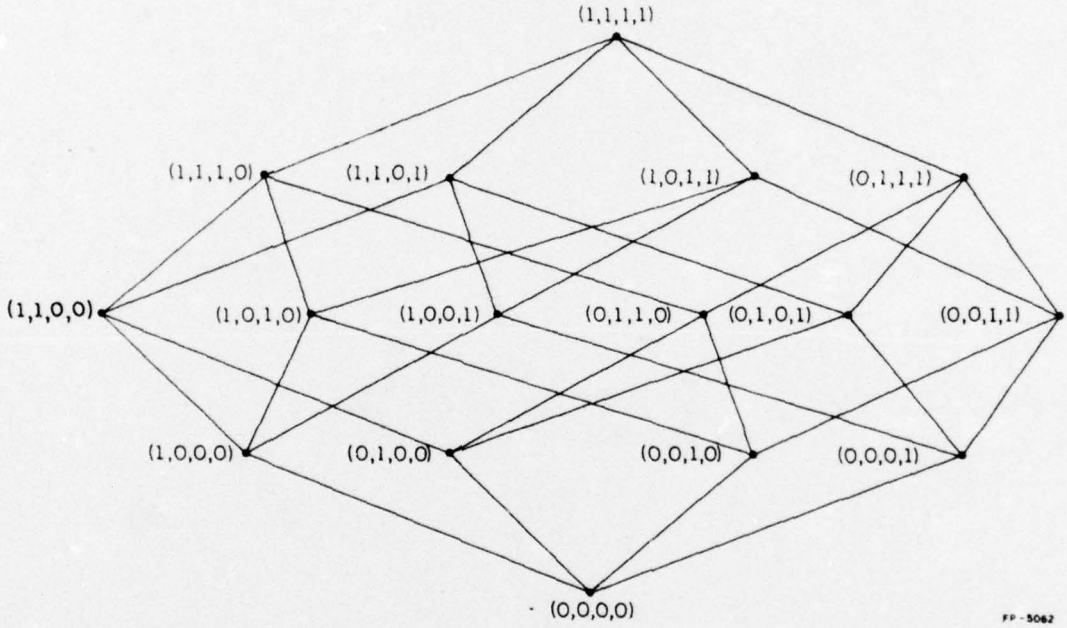


Figure 3.1. The Boolean lattice containing members of the 4-cube.

Definition 3.1: An unordered code of length r is a set of vertices of the r-cube which forms an antichain.

Definition 3.2: A maximal unordered code of length r is a set of vertices of the r-cube which forms a maximal antichain.

We now will see why unordered codes are important when considering unidirectional faults.

Theorem 3.1: If any network, all of whose output lines are potential fault sites, is to be TSC w.r.t. unidirectional faults, then its output code space must be an unordered code.

Proof: Assume the existence of a network G which is TSC w.r.t. unidirectional faults with an output code space containing two members y and z such that $z \leq y$. This means that in each position z has a 1, y also has a 1. From the unidirectional fault assumption, we see that any number of the gates which produce primary outputs of G may simultaneously have their outputs fail to 0.

If y is the correct output vertex under some input x, it can be changed to z by allowing primary output gates to fail to 0 wherever $z_i < y_i$. This is a unidirectional fault which causes the output of G under x to be z instead of y. These means that G is not fault secure, so it cannot be TSC. This contradicts the original assumption that G is TSC. Q.E.D.

There is a very useful relationship between unordered codes and inverter-free circuits which the following theorem demonstrates.

Theorem 3.2: Any function using an unordered input code space can be realized by an inverter-free circuit.

Proof: Any function can be realized using inverters only on primary inputs (e.g. a minimal two-level AND-OR realization). Say we realize the given function with such a circuit. Let A_i be the set of input code vertices with a 0 in position i. If x_i is the input variable taken from the ith position, x_i' can be formed as

$$\sum_{a \in A_i} \left(\prod_{j \mid a_j = 1} x_j \right).$$

Now if any appearance of the inverted x_i in the original circuit is replaced by the inverter-free circuit realizing x_i' for all i, then the resulting circuit is inverter-free.

For all code inputs which result in x_i being a 0, x_i' is a 1 from our construction. If x_i is a 1 for input code word a and x_i' is also 1 under input a, then there must be some input code word b which has 1's only in positions where a has 1's. But this implies that $b < a$. Since it was assumed that the input code space was unordered, this is a contradiction. Therefore, if x_i is a 1 for some input code word, x_i' must be a 0 under the same code word. Thus $x_i' = \bar{x}_i$ and the function realized by the modified inverter-free circuit is the same as that realized by the original circuit.

Q.E.D.

Definition 3.3: An unordered code is closed if complementing any bit of any code word orders it with respect to some other code word.

Most commonly used unordered codes are closed, as we shall see. If we restrict ourselves to closed unordered codes, the following theorem results.

Theorem 3.3: Any network which is TSC with respect to unidirectional faults and which has a closed unordered output code space must be inverter-free.

Proof: Assume that there exists a TSC network G which has a closed unordered output code space and which contains inverters. There must be at least one gate with an inverter (a NAND, NOR or simple inverter) whose output passes through no inverters on the way to the primary outputs $y_1, y_2, \ldots, y_s$. Select one such gate, say a NAND gate; a similar argument to the one which will be given could be used if a NOR gate were selected, and a simple inverter conceptually is a one-input NAND gate.

Since G is TSC with respect to unidirectional faults, it must satisfy Definition 2.3', so for any unidirectional fault F, $\exists a \in A$ such that $G(a,f) \neq G(a,\phi)$. This means that if the fault $\{\ell_1/1\}$ is present on an input to the NAND gate (see Figure 3.2), it must be possible to sensitize the line ℓ_1 to at least one primary output. Assume it can be sensitized to the outputs $y_{i_1}, y_{i_2}, \ldots, y_{i_j}$ under code input a as is shown in Figure 3.2. Since there is only one inverter between ℓ_1 and these outputs, the outputs must be 1 with $\{\ell_1/1\}$ not present and 0 with $\{\ell_1/1\}$ present. If all but one of the affected outputs also become s-a-1 (without loss of generality say y_{i_j} is not s-a-1), then the unidirectional fault $\{\ell_1/1, y_{i_1}/1, \ldots, y_{i_{j-1}}/1\}$ under input a results in only the change of y_{i_j} from 1 to 0 as compared with the fault-free case. Since the code is closed, this noncode output must be ordered with respect to some other code output, say b. This means that b has 1's wherever the erroneous word does and has some additional 1's in positions where the erroneous output word does not, say in positions

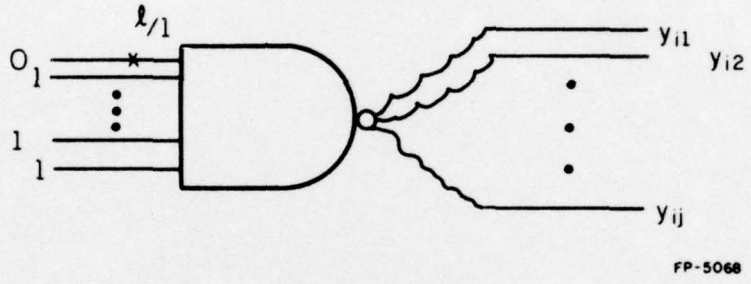


Figure 3.2

$y_{k_1}, y_{k_2}, \ldots, y_{k_\ell}$. Now, the unidirectional fault $\{\ell_1/1, y_{i_1}, y_{i_2}/1, \ldots, y_{i_{j-1}}/1, y_{k_1}/1, y_{k_2}/1, \ldots, y_{k_\ell}/1\}$ under input a results in the incorrect code output b. Thus, G is not TSC with respect to unidirectional faults. This contradiction completes the proof. Q.E.D.

In combination, Theorems 3.1, 3.2, and 3.3 offer strong justification for considering only unordered codes and inverter-free circuits when designing circuits which are TSC with respect to unidirectional faults. Therefore, we will restrict ourselves to inverter-free circuits and unordered codes in the remainder of this chapter. The results of the next section will provide additional justification for these restrictions.

Two lemmas without proofs will now be presented. These lemmas will be useful in later sections and proofs can be found in [23] and [24].

Lemma 3.1: In an inverter-free network G if input $y \leq x$ then $G(y,\phi) \leq G(x,\phi)$.

Lemma 3.2: In an inverter-free network G if f_o is a unidirectional s-a-0 fault and f_1 is a unidirectional s-a-1 fault then

$$\forall x \in X \quad G(x,f_o) \leq G(x,\phi) \qquad \text{and}$$

$$\forall x \in X \quad G(x,\phi) \leq G(x,f_1).$$

3.3. Construction of Arbitrary TSC Functional Blocks

We will now show that any mapping of an unordered code onto an unordered code can be realized by a network which is TSC for unidirectional faults. The method for doing this is surprisingly easy. The connection of

these TSC blocks to form TSC and E-TSC networks will then be discussed. We begin with an important theorem.

Theorem 3.4: Any inverter-free network with an unordered output code space is P-FS with respect to unidirectional faults.

Proof: Since there are no inverters, the parity of inversions from any fault site to any output is obviously even. This means that the sensitization set for a unidirectional fault set can just contain vectors with 1's and n's only or with 0's and n's only. If the ⓔ operator is applied to any of these sensitization vectors and a code word, the resulting error vector is ordered with respect to the code word. Since the code is unordered, the error vector either must be the correct code word or must be a noncode word. Q.E.D.

Corollary 3.1: Any inverter-free network with an unordered output code space is E-TSC with respect to unidirectional faults.

Proof: From Theorems 2.3 and 3.4.

Corollary 3.2: Any mapping from an unordered code space onto an unordered code space can be made TSC with respect to unidirectional faults.

Proof: From Theorem 3.2, such a mapping can always be realized without inverters. Theorem 3.4 states that this mapping is P-FS, and reducing the circuit with respect to any undetected faults as discussed following Theorem 2.5 yields a TSC circuit. Q.E.D.

Having demonstrated the ease with which functional blocks can be constructed such that they are TSC with respect to unidirectional faults, we will turn to the interconnection of these blocks to form larger TSC networks.

Lemma 3.2 states that a unidirectional fault can only produce unidirectional output errors. On the other hand, Lemma 3.1 states that if a noncode input word is received which contains a unidirectional error, the output is either the correct code output or the code output corrupted by a unidirectional error. Thus, any inverter-free TSC network with unordered input and output code spaces is error secure with respect to unidirectional errors--just the type of error which this type of network produces under unidirectional faults.

Since any interconnection of inverter-free blocks is also inverter-free, the next theorem can be easily proved.

Theorem 3.5: Any interconnection of inverter-free functional blocks which have unordered input and output code spaces is P-FS with respect to unidirectional faults.

Proof: Follows directly from Theorem 3.4.

We should note that the interconnection is not only P-FS with respect to the union of the fault sets of the component blocks, but is P-FS with respect to all unidirectional faults when the interconnection is treated as a whole.

Theorem 3.5 provides a very useful result: it allows us to take inverter-free functional blocks which have unordered input and output code spaces "off the shelf," and to interconnect them to get a network which meets the desired goal outlined in Section 2.1.

3.4. Error Preserving TSC Functional Blocks

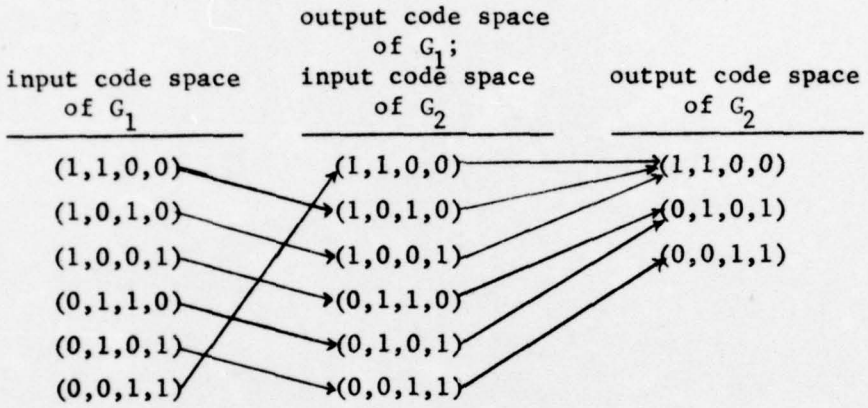
Although some rather strong results concerning TSC circuits were developed in the last section, a major problem remains--the design of functional blocks which are not only error secure but which are error transmitting.

Error transmitting circuits have several applications. Perhaps the most important is in the construction of TSC checkers which have been shown to be very useful.

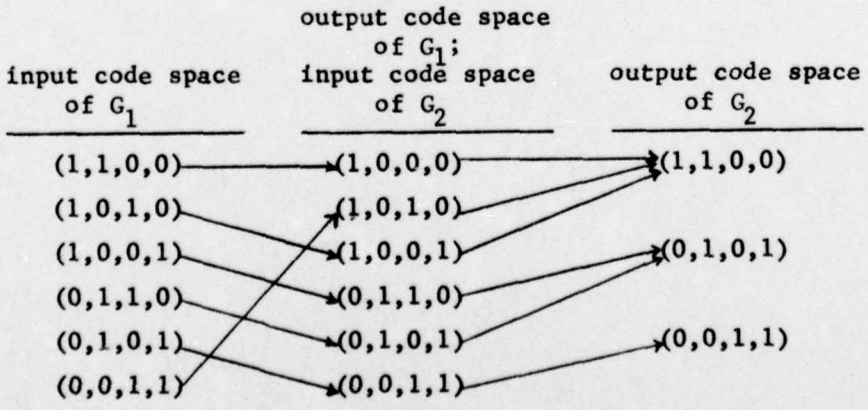
A second application of error transmitting functional blocks is in the combinational part of TSC sequential machines [41]. Here, a circuit which is merely error secure could cause the machine to leave an error state which has been reached due to a fault and return the machine to the correct state. While we continue to get correct outputs, the detection of the fault is prolonged and may require a sequence of transitions to detect it rather than a single transition.

A third reason we may desire error transmitting functional blocks is to facilitate fault location after an indication of a failure has been received. As pointed out in the previous section, if error secure functional blocks are interconnected, a P-FS (and E-TSC) functional block results. However, nondetectable failures can cause a code space between modules to be changed. If this happens and a later detectable failure occurs, an apparent noncode word at the output of a block and a code word at its input do not necessarily mean that a failure within the block caused the erroneous output. To prevent such an occurrence, it is desirable to have interconnections which are TSC. According to Theorem 2.1, error transmitting blocks make this possible.

Example 3.2: Figure 3.3 shows the connection of two functional blocks, G_1 and G_2. G_1 and G_2 perform the following mappings of code space inputs:



The two blocks shown in Figure 3.3 which realize these mappings are individually TSC. However, when they are connected, the resulting circuit is E-TSC but not TSC since fault $\{\ell_1/0\}$ (among others) is not detectable with code space inputs. The circuit reduced with respect to $\{\ell_1/0\}$ performs the following mapping:



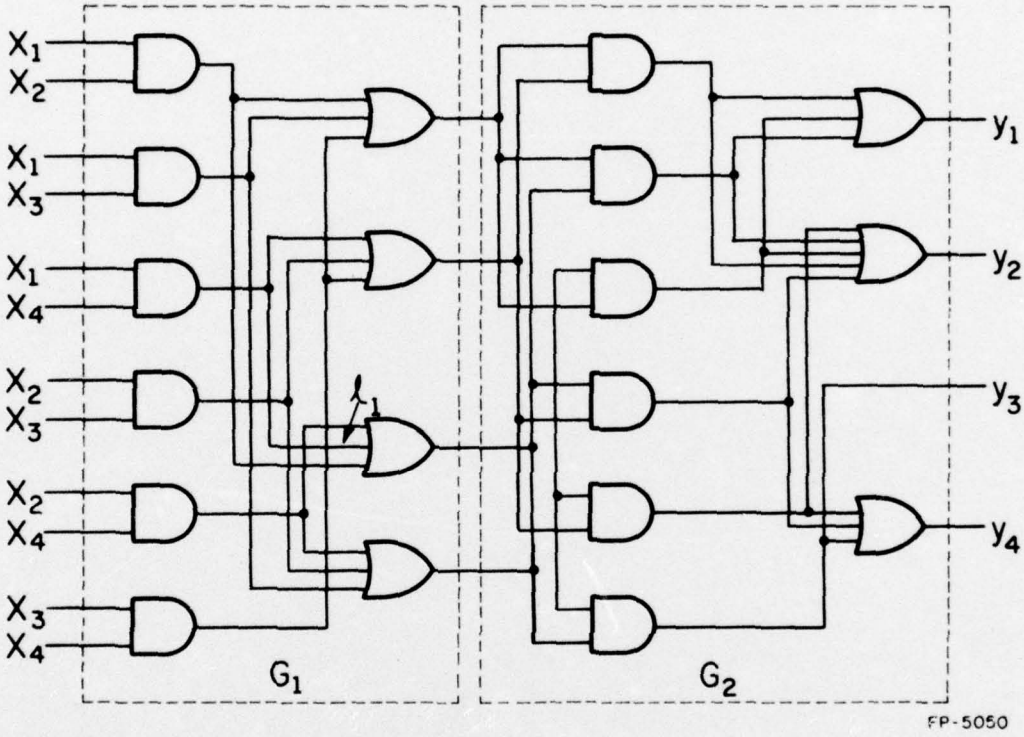


Figure 3.3. Interconnection of functional blocks G_1 and G_2 from Example 3.2.

Thus, the intermediate code space has been changed while the connection as a whole still works properly. □

As has been pointed out, due to Lemmas 3.1 and 3.2, in inverter-free circuits any noncode word generated by assumed unidirectional faults must be ordered with respect to the correct code output. That is, for a functional block G, the potential error set $\mathcal{E} \subseteq \{\text{all unidirectional errors}\}$. Thus, to produce a block G that is error transmitting in an inverter-free interconnection, by Definition 2.9 it is necessary to make G error preserving with respect to unidirectional errors, or as we will abbreviate it "u-error preserving."

In this section we will present general conditions which, if met by a mapping of an unordered code, imply that any inverter-free circuit realizing that mapping must be u-error preserving. Furthermore, if, for some reason, a code disjoint mapping is desired, one particular realization will be shown to be code disjoint. We will then specialize these results to particular codes, and show how u-error preserving realizations for mappings which do not meet the conditions might be constructed.

We now present a main theorem. The theorem refers to code vertices as they appear in the Boolean lattice.

Theorem 3.6: In a mapping of an unordered code onto an unordered code, if each input noncode vertex which is adjacent to an input code vertex a is ordered with respect to at least one other input code vertex b, so that a and b map to different outputs, then any inverter-free realization of the mapping will be u-error preserving.

Proof: Let G be an inverter-free functional block that realizes the given mapping. Consider any noncode word x_1 which results from a

unidirectional error; then $x_1 < a$ or $a < x_1$ for some code word a. We will consider the first case only, since a nearly identical argument can be used for the second.

Due to transitivity of the covering relation, there exists a non-code word x_2 which is adjacent to a such that $x_1 \leq x_2 < a$. We know that x_2 is also ordered with respect to another code word b such that $G(a,\phi) \neq G(b,\phi)$, and since a and b are unordered, $x_2 < b$.

Due to Lemma 3.1, $G(x_2,\phi) \leq G(a,\phi)$ and $G(x_2,\phi) \leq G(b,\phi)$. Furthermore, since the output code space is unordered $G(a,\phi) \not\leq G(b,\phi)$ and $G(b,\phi) \not\leq G(a,\phi)$. In order for $G(x_2,\phi)$ to equal $G(a,\phi)$ it would be necessary for $G(a,\phi) \leq G(b,\phi)$ which is a contradiction, therefore $G(x_2,\phi) < G(a,\phi)$, and $G(x_2,\phi)$ must be a noncode word, and $G(x_1,\phi)$ is a noncode word by Lemma 3.1.

If the unidirectional error results in $a < x_1$, we get the same result. Therefore G is u-error preserving. Q.E.D.

In order to construct a u-error preserving TSC functional block when the mapping satisfies Theorem 3.6, we should first construct any inverter-free realization of the mapping. By Theorem 3.4, the circuit must also be P-FS. The circuit can then be reduced with respect to any faults which are not detected by the input code space, and the final circuit must be TSC as well as being u-error preserving.

Corollary 3.3: If the input code space of Theorem 3.6 is maximal, then any inverter-free realization of the mapping will be code disjoint.

Proof: In a maximal code every noncode space vertex is ordered with respect to at least one code word. Thus, all noncode input vertices map to a noncode space output vertex. Q.E.D.

Not all input codes and not all mappings meet the conditions of Theorem 3.6 and Corollary 3.3. The class of input codes which can potentially meet the conditions is the class of closed unordered codes. This is true because any noncode word which is adjacent to a code word can be generated by complementing a bit in the code word. In order for the noncode word to always be ordered with respect to a different code word, the code must be closed.

Corollary 3.4: Any inverter-free realization of a one-to-one mapping from a closed unordered code onto an unordered code must be u-error preserving.

Proof: Obvious from the preceding discussion and Theorem 3.6.

Corollary 3.5: If the condition of Theorem 3.6 is met by a mapping of an unordered code onto an unordered code then there exists at least one TSC network realizing the mapping which is code disjoint.

Proof: (by construction). An obvious two-level realization of the mapping can be formed as follows.

1) Form an AND gate for each input code vertex with inputs corresponding to coordinates of the vertex which are 1's.

2) If the output space contains vertices of length s, form s OR gates, each of which corresponds to one output bit. The inputs of each OR gate are the outputs of all the AND gates that correspond to code inputs which map to code outputs for which the coordinate of the vertex which the OR gate represents is a 1.

The circuit constructed in this way will realize the desired mapping, and since no inverters are used it must be P-FS and, therefore,

fault secure. It is easily verified that all unidirectional faults can be tested by code inputs, so the circuit is TSC.

By Theorem 3.6, any noncode input vertex which is ordered with respect to a code vertex will be mapped to a noncode output vertex.

If a noncode vertex is not ordered with respect to any code vertex, all of the AND gates will output a 0, so the output vertex will be all 0's which must be a noncode vertex in a closed unordered code. Thus, all noncode input vertices map to noncode output vertices, and the functional block is code disjoint. Q.E.D.

Only a relatively small number of mappings satisfies the conditions of Theorem 3.6 and can be immediately realized by a u-error preserving functional block. However, by forming a serial decomposition of a mapping, many more can be realized by u-error preserving circuits.

A serial decomposition of a mapping results in a serial interconnection of functional blocks which realize the desired mapping. Such a serial interconnection may appear to be a special case of the interconnections discussed in Chapter 2. However, these serial interconnections must be u-error preserving in addition to being TSC. The interconnections discussed in Chapter 2 resulted in networks where any fault contained stuck lines which were all located in the same block. The serial interconnections presented here will allow unidirectional faults containing stuck lines in any number of blocks in the interconnection.

Theorem 3.7: A serial connection of inverter-free functional blocks $G_1, G_2, \ldots, G_n$ (see Figure 3.4) is TSC with respect to unidirectional faults and is u-error preserving if

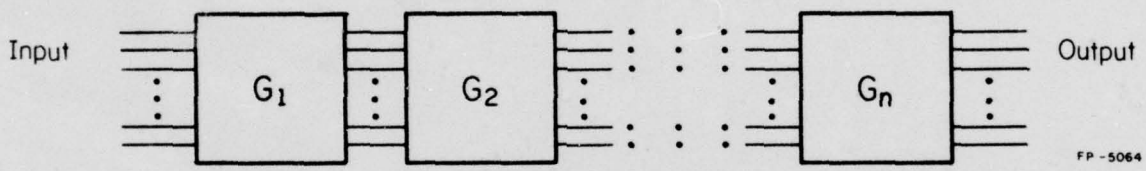


Figure 3.4. A serial interconnection of functional blocks.

1) $\forall i$ G_i is TSC with respect to unidirectional faults;

2) $\forall i$ G_i is u-error preserving;

3) $\forall i$ G_i is sufficiently exercised.

Proof: Since the interconnection contains no inverter, it is P-FS and fault secure.

If a unidirectional fault, f, is present, assume G_j is the first block containing any component of the fault. Since G_j is sufficiently tested, G_j receives some input a, such that $G_j(a,f) < G_j(a,\phi)$ if f is a s-a-0 fault and $G_j(a,\phi) < G_j(a,f)$ if f is a s-a-1 fault due to Lemma 3.2. All subsequent blocks which are fault-free and receive a noncode input which covers (is covered by) a code input pass on a noncode output which covers (is covered by) a code output due to the u-error preserving property and Lemma 3.1. In addition, any block which contains a component of f that receives a noncode input which covers (is covered by) a code input passes on a noncode output which covers (is covered by) a code output due to the u-error preserving property and Lemmas 3.1 and 3.2. Thus, the error created at the output of G_j is propagated to the output of the interconnection. Therefore, the interconnection is TSC with respect to unidirectional faults.

Due to the u-error preserving property of each block, the interconnection is also u-error preserving. Q.E.D.

Corollary 3.6: A serial interconnection of inverter-free functional blocks $G_1, G_2, \ldots, G_n$ is TSC with respect to unidirectional faults and is code disjoint if

1) $\forall i$ G_i is TSC with respect to unidirectional faults;

2) G_1 is code disjoint;

3) $\forall i$, $i > 1$, G_i is u-error preserving;

4) $\forall i$, G_i is sufficiently exercised.

Proof: Similar to the proof of Theorem 3.7.

Although serial decompositions will allow many mappings of closed codes to be TSC and u-error preserving, some mappings of closed codes cannot be made TSC and u-error preserving as the next theorem will show.

Theorem 3.8: A mapping of a closed unordered code onto a closed unordered code cannot be realized such that the network is TSC with respect to unidirectional faults and is u-error preserving if all of the members of the input code space except for one map to the same member of the output code space and if the mapping is not one-to-one.

Proof: The only closed unordered code with two members is the 1-out-of-2 code; for this reason the output code space must be the 1-out-of-2 code. Due to Theorem 3.3, the circuit must be inverter-free.

Let the two outputs be y_1 and y_2. Without loss of generality assume that only one input code word maps to $(y_1,y_2) = (0,1)$. Then all the others map to $(y_1,y_2) = (1,0)$. Then y_1 must be realized as the output of an OR gate and y_2 must be realized as the output of an AND gate, since testing an OR gate for single faults requires one test with a 0 output and a test with a 1 output for each input of the gate, and testing an AND gate requires only one test with a 1 output and possibly many with a 0 output. If no gates are used to form y_1 or y_2 we conceptually have a single input gate.

It is easy to verify that a single AND gate and a single OR gate are not sufficient. The AND gate must be fed by an OR gate or the OR gate must be fed by an AND gate. We will examine the latter case; the former can be handled with a similar argument.

To test the AND gate for single faults there must be at least two tests which make its output a 0 and one which makes its output a 1. All of the tests cannot be observed at y_1 since this would require y_1 to be 0 for two tests. This means the AND gate must fan out into the network forming y_2. However, it cannot feed one of the inputs of the AND gate forming y_2 since testing the output y_2 s-a-0 would then require both y_1 and y_2 to be 1. For this reason, it must feed an OR gate which is in the network forming y_2. But, to test this gate, it must fan out into the circuit forming y_1. Furthermore, it requires an additional AND gate in y_1, which in turn must fan out to y_2 and which requires an additional OR gate in y_2. We see that this process can never terminate. Therefore, such a circuit cannot be designed which is TSC with respect to single faults which are a subset of unidirectional faults. Q.E.D.

3.4.1. TSC Circuits Using m-out-of-n Codes

Using the tools we have developed in the preceding section, we will now turn to the design of some specific TSC u-error preserving functional blocks. The designs to be given are intended to demonstrate the utility of the theorems. However, they have been chosen to give useful designs of more common circuits.

We begin by considering mappings of m-out-of-n codes. An m-out-of-n code is the set of all m-vertices of the n-cube. m-out-of-n codes are closed maximal unordered codes.

Since m-out-of-n codes are closed, Theorem 3.6 applies to them. The mapping of a code space onto a code space partitions the input code into

blocks where all members of the same block map to the same output word. Using this notion of a partition induced by a mapping, we will restate Theorem 3.6 as a corollary which has been specialized for m-out-of-n codes.

Corollary 3.7: In a mapping of an m-out-of-n code onto an unordered code, if

1) each (m+1)-vertex covers some code vertex in at least two blocks of the partition induced by the mapping;
2) each (m-1)-vertex is covered by some code vertex in at least two blocks of the partition induced by the mapping,

then any inverter-free realization of the mapping is u-error preserving.

Proof: This is an obvious restriction of Theorem 3.6 to m-out-of-n codes.

A well known theorem in combinatorics, Ramsey's theorem [25], can be shown to be applicable to the partition of m-out-of-n code vertices. From this theorem and Ramsey numbers [25] which are suggested by the theorem, we can derive many useful facts concerning u-error preserving functional blocks, particularly in the area of TSC checkers for m-out-of-n codes.

In [26] a generalized form of Ramsey's theorem is presented in terms of subsets of a set, and the Ramsey numbers $N(p_1,p_2,\ldots,p_t;r)$ are defined. We present here a version in terms of vertices of the n-cube which is specialized for our purposes. Let $R(t,m) = N(\underbrace{m+1,m+1,\ldots,m+1}_{t};m)$.

Theorem 3.9: (Ramsey) For any given integers $m \geq 1$ and $t \geq 2$ there exists a finite number $R(t,m)$ depending solely on m and t such that for any n-cube $C, n \geq R(t,m)$, and for any arbitrary partition of all the

m-vertices of C into t blocks, there exists an m+1-vertex of C which covers only m-vertices in one of the blocks.

The Ramsey numbers of the form R(t,1) are easily computed, however if $m \geq 2$ their determination is much more difficult and very few are known [26].

Conditions 1 and 2 of Corollary 3.7 are in some sense duals. That is, if a satisfactory partition exists for an m-out-of-n code, a satisfactory partition for the (n-m)-out-of-n code can be determined by simply complementing the partitioned m-vertices. For this reason, we will concentrate on the case where $m \leq \left\lfloor \frac{n}{2} \right\rfloor$.

Let $P^+(t,m,n)$ be the collection of partitions of the m-vertices of the n-cube into t blocks which satisfy Condition 1 of Corollary 3.7. Let $P^-(t,m,n)$ be the collection of partitions of the m-vertices of the n-cube into t blocks which satisfy Condition 2 of Corollary 3.7. Let $p_1|p_2|\ldots|p_t$ denote a partition of the m-vertices where $p_1,p_2,\ldots,p_t$ are the t blocks of the partition. If $P^-(t,m,n)$ is not empty, an arbitrary partition $p_1|p_2|\ldots|p_t$ can obviously be converted into a member of $P^-(t,m,n)$ by moving some subset of p_1 into $p_2,p_3,\ldots,p_t$, some subset of p_2 into $p_1,p_3,\ldots,p_t$ etc. We will perform such moves, but only one vertex at a time, and only if a certain condition is met.

Definition 3.4: A move of a vertex from p_i into p_j is a proper move only if it covers an (m-1)-vertex which was not previously covered by any member of p_j.

Lemma 3.3: Starting with an arbitrary initial partition $p_1|p_2|\ldots|p_t$ of the m-vertices, some series of proper moves will lead to a member of $P^-(t,m,n)$ if $P^-(t,m,n) \neq \emptyset$.

Proof: Say $p_1'|p_2'|\ldots|p_t' \in P^-(t,m,n)$ and $p_1|p_2|\ldots|p_t$ can be converted to $p_1'|p_2'|\ldots|p_t'$ by moving $p_i^j \subseteq p_i$, $i \neq j$ into p_j. We see that $p_j' = (p_j - \bigcup_{\substack{i=1 \\ i\neq j}}^{t} p_j^i) \bigcup_{\substack{i=1 \\ i\neq j}}^{t} p_i^j$. Now, we will make a series of proper moves from one block of the partition into another. However, only members of p_i^j will be moved into p_j for any j. Assume we reach a partition $p_1''|p_2''|\ldots|p_t''$ such that we can no longer make a proper move.

If $p_1''|p_2''|\ldots|p_t'' \in P^-(t,m,n)$ we are done. If not, then there is some (m-1)-vertex, q, covered by only one p_i''. Without loss of generality, assume q is an (m-1)-vertex covered only by members of p_k''.

Since $p_1'|p_2'|\ldots|p_t' \in P^-(t,m,n)$, then some member of p_k'' which covers q is not in p_k'. Assume it is in p_ℓ' (and in p_k^ℓ). Moving the member of p_k'' which covers q into p_ℓ'' is a proper move; hence, we have a contradiction. Therefore, $p_1''|p_2''|\ldots|p_t''$ must be a member of $P^-(t,m,n)$. Q.E.D.

Lemma 3.4: Given any member of $P^+(t,m,n)$, after a proper move the resulting partition will also be a member of $P^+(t,m,n)$.

Proof: Let $p_1|p_2|\ldots|p_t$ be a member of $P^+(t,m,n)$. Without loss of generality, assume a proper move is made from p_1 into p_2 which results in a partition $p_1'|p_2'|\ldots|p_t'$, and that the moved m-vertex is intended to cover the (m-1)-vertex $(\underbrace{1,1,1,\ldots,1}_{(m-1)\ 1\text{'s}},0,0,\ldots,0)$. Then, before the move, the m-vertices

$$
\begin{array}{l}
\overbrace{(1,1,1,\ldots,1}^{\text{(m-1) 1's}},1,0,0,\ldots,0,0) \\
(1,1,1,\ldots,1,0,1,0,\ldots,0,0) \\
(1,1,1,\ldots,1,0,0,1,\ldots,0,0) \\
\quad\vdots \qquad\qquad\qquad\quad \vdots \\
(1,1,1,\ldots,1,0,0,0,\ldots,0,1)
\end{array}
$$

must all have been in p_1. Each (m+1)-vertex which covers one of these m-vertices also covers another. Thus, if any one of these m-vertices is selected for a proper move, all the (m+1)-vertices which cover it still cover other members of p_1. Therefore if $p_1|p_2|\ldots|p_t \in P^+(t,m,n)$ before the proper move $p_1'|p_2'|p_3|\ldots|p_t \in P^+(t,m,n)$ after the proper move. Q.E.D.

Lemma 3.5: *If* $P^+(t,m,n) \neq \emptyset$ *and* $P^-(t,m,n) \neq \emptyset$. *Then* $P^+(t,m,n) \cap P^-(t,m,n) \neq \emptyset$.

Proof: Given a member of $P^+(t,m,n)$ initially, by Lemma 3.3 some set of proper moves will lead to a member of $P^-(t,m,n)$ since $P^-(t,m,n) \neq \emptyset$. Because each move was proper, by Lemma 3.4 the resulting member of $P^-(t,m,n)$ must also be a member of $P^+(t,m,n)$. Q.E.D.

Theorem 3.10: For $m < \left\lfloor \frac{n}{2} \right\rfloor$, $R(t,m) > n$ iff the m-vertices of the n-cube can be partitioned to meet Conditions 1 and 2 of Corollary 3.7, and for $m \geq \left\lfloor \frac{n}{2} \right\rfloor$, $R(t,n-m) > n$ iff the m-vertices of the n-cube can be partitioned to meet Conditions 1 and 2 of Corollary 3.7.

Proof: If $R(t,m) > n$ then the m-vertices of the n-cube can be partitioned into t blocks such that all the m-vertices covered by an (m+1)-vertex are not in the same block. This implies $P^+(t,m,n) \neq \emptyset$.

Since $n-m \geq m$ for $m \leq \left\lfloor \frac{n}{2} \right\rfloor$, $R(t,n-m) > n$. Therefore, $P^{+}(t,n-m,n) \neq \phi$, and $P^{-}(t,m,n) \neq \phi$.

$P^{+}(t,m,n) \cap P^{-}(t,m,n) \neq \phi$ by Lemma 3.5, which means that a partition of m-vertices can be selected such that Conditions 1 and 2 are satisfied.

Satisfying Condition 1 implies that $P^{+}(t,m,n) \neq \phi$ which implies that $R(t,m) > n$.

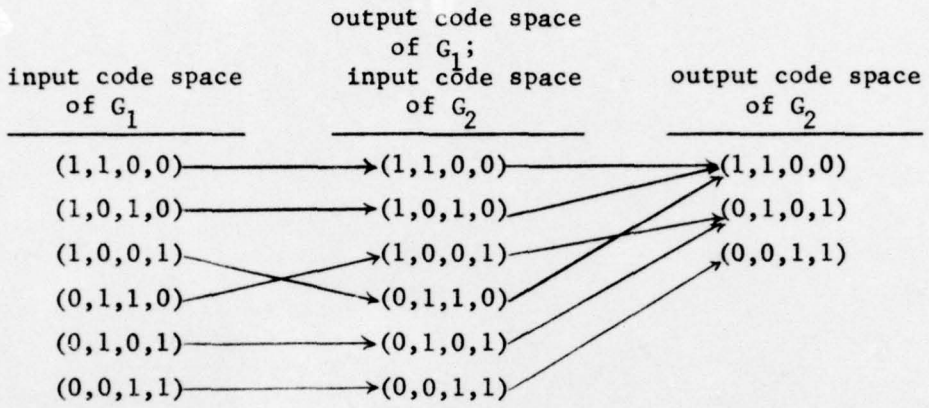
By a similar argument we can prove the case where $m \geq \left\lceil \frac{n}{2} \right\rceil$. Q.E.D.

Theorem 3.10 is a very useful theorem; for example, it is easy to show that $R(t,1) = t+1$. Therefore, if we have a 1-out-of-n code, $R(t,1) > n$ only if $t \geq m$ which means that the only mapping of a 1-out-of-n code which satisfies Theorem 3.6 is a one-to-one mapping. As another less trivial example, since $R(3,2) = 17$ [26], we can observe that a 2-out-of-n code with $n \geq 17$ cannot be mapped onto an output code with 3 members and still satisfy Theorem 3.6 regardless of the particular mapping.

There is also a rather negative result attached to Theorem 3.10--the Ramsey numbers are very difficult to compute in general. Numbers of the form $R(t,m)$ have been found only for $m = 1$; $t = 2$, $m = 2$, and $t = 3$, $m = 2$. Considering the fact that the Ramsey number problem is a well-known problem in combinatorics, this implies that the structure of m-out-of-n codes is actually quite complex since such a seemingly easy problem like the partition of the vertices as described in Corollary 3.7 is actually very difficult (and thus far unsolved) in many cases.

Example 3.3: Suppose we wish to realize functional block G of Example 3.2 so that it is u-error preserving. We see that (1,0,0,0) is covered only by code vertices which map to (1,1,0,0); therefore, the

conditions of Corollary 3.7 are not satisfied. However, G can be serially decomposed into two blocks which each satisfy the conditions of Theorem 3.7:



The mapping specified by G_1 is 1 to 1 and satisfies Corollary 3.4; the mapping specified by G_2 now satisfies the conditions of Corollary 3.7. Thus, TSC realizations of G_1 and G_2 lead to a TSC u-error preserving realization of G by Theorem 3.7. □

3.4.2. The Construction of m-out-of-n TSC Checkers

For the unidirectional fault set, any m-out-of-n checker which is to be TSC must have at least two outputs [6]. The designs for m-out-of-n TSC checkers given here will all use two outputs except for the 1-out-of-3 and 2-out-of-3 checkers which must have more than two outputs to agree with Theorem 3.8. Checkers by definition must be error transmitting. Presumably, their potential error sets are all unidirectional errors so we need to design them to be u-error preserving to get the error transmitting property when they are used. However, we should note that in the case of m-out-of-n codes which

are maximal codes, any u-error preserving realization is also code disjoint.

Since we desire two-output checkers, then their output code spaces must be the 1-out-of-2 code since this is the only unordered code of length 2, and the output code must be unordered according to Theorem 3.1.

The restriction of Theorem 3.6 to these two-output checkers leads to the following corollary.

Corollary 3.8: In a mapping of an m-out-of-n code onto a 1-out-of-2 code if

1) each (m+1)-vertex covers some m-vertex mapping to (1,0) and some m-vertex mapping to (0,1) and
2) each (m-1)-vertex is covered by some m-vertex mapping to (1,0) and some m-vertex mapping to (0,1)

then any inverter-free functional block realizing the mapping is u-error preserving.

Proof: This is a special case of Theorem 3.6.

By using Corollary 3.8 and reducing the resulting inverter-free circuit with respect to any undetected unidirectional faults, a TSC checker can be constructed. Since Conditions 1)and 2) of Corollary 3.8 are sufficient for any inverter-free realization to be u-error preserving, they are obviously sufficient for a two-level realization. We will now show that they are necessary for a two-level realization.

Lemma 3.6: In a two-level inverter-free TSC checker G with outputs y_1 and y_2, the only fanout which can occur is at the primary inputs.

Proof: By exhausting all possible forms of fanout (except at the primary inputs) it can easily be shown that each form leads to a circuit which either cannot possibly be a checker (is not u-error preserving) or which is not self-testing. The only form of fanout which even deserves consideration is the fanout from an AND (OR) gate output of an AND-OR (OR-AND) realization of one of the two outputs y_1 into the output gate which forms y_2. In this case, a s-a-0 (s-a-1) fault where the fanout line enters the output gate forming y_2 cannot be tested with a code input. Q.E.D.

Lemma 3.7: In a two-level inverter-free TSC checker G with outputs y_1 and y_2, an AND-OR realization of y_1 or y_2 must have exactly m inputs to each AND gate, and an OR-AND realization of y_1 or y_2 must have exactly n-m inputs to each OR gate.

Proof: In an AND-OR realization of y_1 or y_2, if an AND gate has fewer than m inputs, it will produce a value 1 for a noncode vertex, say z, which has fewer than m 1's. This means that the checker must output $(y_1,y_2) = (1,1)$ for z. However, for any code vertex which covers z to map to $(y_1,y_2) = (0,1)$ or (1,0) Lemma 3.1 must be violated.

If an AND gate in an AND-OR realization of y_1 or y_2 has more than m inputs, then it will require a vertex with more than m 1's to check the output of the AND gate s-a-0. Therefore, such a circuit cannot be self-testing with respect to unidirectional faults.

Thus, every AND gate in an AND-OR realization must have exactly m inputs. By a dual argument, it can be proved that every OR gate in an OR-AND realization must have exactly n-m inputs. Q.E.D.

According to Lemma 3.7, each AND gate in an AND-OR realization of y_1 or y_2 corresponds to exactly one code vertex and has inputs corresponding to the coordinates of the vertex which are 1's. The AND gate produces the value 1 when the code vertex it corresponds to is present at the checker inputs and the value 0 when any other code vertex is present.

Each OR gate in an OR-AND realization of y_1 or y_2 also corresponds to exactly one code vertex, and has inputs which correspond to the coordinates of the vertex which are 0's. The OR gate produces the value 0 when the code vertex it corresponds to is present at the checker inputs, and the value 1 when any other code vertex is present.

Theorem 3.11: For a two-level inverter-free functional block G realizing a mapping of an m-out-of-n code onto the 1-out-of-2 code to be u-error preserving, Conditions 1) and 2) of Corollary 3.8 must be met.

Proof: Assume that Condition 1) is not satisfied. Then, without loss of generality, assume there is no m-vertex covered by a particular (m+1)-vertex x which maps to (1,0).

Case 1: y_1 is realized by an AND-OR circuit. Then $G(x,\phi) = (1,1)$ to satisfy Lemma 3.1; therefore, there must be an AND gate in the circuit forming y_1 which goes to 1 for x but which is 0 for any code vertex. This requires an AND gate with more than m inputs which violates Lemma 3.7.

Case 2: y_1 is realized by an OR-AND circuit. Then y_1 cannot have as a true vertex any of the m-vertices covered by x, and all of the true vertices of y_1 must have at least one 1 in the n-(m+1) positions which are not covered by x. This implies that y_1 is 1 for the (m-1)-vertex formed by

complementing x. This means that $(y_1,y_2) = (1,1)$ for an (m-1)-vertex. This leads to a contradiction of Lemma 3.1.

So, if Condition 1 is not satisfied, then the mapping cannot be realized by a two-level TSC checker. If a TSC checker cannot be realized, then a u-error preserving mapping cannot be realized either since it could be reduced to a TSC checker.

Now, assume Condition 2 is not satisfied. Without loss of generality assume all m-vertices covering a particular (m-1)-vertex x map to (1,0).

Case 1: y_1 is realized by an AND-OR circuit. Consider an AND gate which corresponds to an m-vertex covering x. To test the input which does not correspond to one of the m-1 coordinates of x for a s-a-1 fault, it is necessary to make each of the m-1 inputs a 1 and the line being tested a 0. The self-testing property requires that this can be done by an m-vertex which means that one more input variable must be set to 1. Whichever variable is chosen, the vertex must cover x, so it must map to (1,0) and thus desensitize the path from the fault site being tested. Therefore, the circuit cannot be self-testing.

Case 2: y_1 is realized by an OR-AND circuit. To avoid violating Lemma 3.1 $G(x,\phi) = (0,0)$. But for any m-vertex which covers x, the output must be (1,0). This requires an OR gate with more than n-m inputs which contradicts Lemma 3.7.

Therefore, if Condition 2 is not met, a TSC checker cannot be realized by a two-level inverter-free circuit, so a u-error preserving realization of the mapping does not exist. Q.E.D.

As we observed earlier, the particular code space output onto which a code space input is mapped by a TSC checker is unimportant. For this reason the mapping implied by any partition of code words which satisfies the conditions of Corollary 3.8 will lead to a TSC checker. Thus, the importance of Theorem 3.10 is that it gives a relation which must exist between m and n in order for a satisfactory partition to exist.

In the case of TSC checkers, Theorem 3.10 leads us to examine Ramsey numbers of the form R(2,m). The only known Ramsey numbers of this kind are R(2,1) and R(2,2) which are 3 and 6 respectively [26]. From this information we can deduce that the 1-out-of-2 code is the only 1-out-of-n code which satisfies Corollary 3.8 and the 2-out-of-4 and 2-out-of-5 codes are the only 2-out-of-n codes which can satisfy Corollary 3.8. Since a satisfactory partition for an m-out-of-n code implies there is a satisfactory partition for the (n-m)-out-of-n code, we can also deduce that the 3-out-of-5 code can be partitioned to satisfy Corollary 3.8. From these observations we can conclude such things as the nonexistence of a two-level 2-out-of-6 TSC checker.

It has been shown that $13 \leq R(2,3) \leq 17$ [27,28] which allows us to conclude that partitions exist which satisfy Corollary 3.8 for 3-out-of-n codes for $6 \leq n \leq 12$, that they may exist for 3-out-of-n codes for $13 \leq n \leq 16$, and that no satisfactory partitions exist for 3-out-of-n codes for $17 \leq n$, and we can reach similar conclusions about (n-3)-out-of-n codes. Bounds for $R(2,n)$ $n>3$ can also be determined [26]; however, they are very loose.

From a consideration of coverings, it appears to be easy to partition the m-out-of-n code words for $m = \left\lfloor \frac{n}{2} \right\rfloor$, $m > 1$ so that the conditions of Corollary 3.8 are met, and that a relatively large number of such partitions exists.

As m becomes larger (or smaller) for a fixed n, fewer and fewer satisfactory partitions exist and it becomes increasingly difficult to find one. Finally, when $R(2,m) \leq n$ it becomes impossible to find a satisfactory partition.

An m-out-of-n code has an optimal or a near optimal number of code words when $m \simeq \left\lfloor \frac{n}{2} \right\rfloor$ [29]. For this reason, m-out-of-n codes with $m \simeq \left\lfloor \frac{n}{2} \right\rfloor$ are of the greatest practical interest. For these particular codes, the designer of totally self-checking checkers is given a great deal of latitude in choosing a design which fits his particular constraints. Any of a large number of partitions of the m-vertices may be used, and given the partition, any inverter-free circuit may be used to realize the checker. Thus, a designer may find that one partition and a particular circuit give him a design with minimal gates, lines, or whatever property he may be interested in. Therefore, although the problem of finding a satisfactory partition of the code vertices is in general difficult, it is relatively easy for codes of practical interest.

Having shown necessary and sufficient conditions for the construction of two-level TSC checkers, we will now show how two-level TSC checkers with a minimal number of gates can be constructed (provided m and n are sufficient for one to exist).

Let P_1 and P_2 be the two blocks of the partition of m-out-of-n code words which satisfies Corollary 3.8. For any member a of P_1, $G(a,\phi) = (1,0)$, and for any member b of P_2, $G(b,\phi) = (0,1)$. If a satisfactory partition of the m-vertices of the n-cube exists, then H(m,n) is the minimal number of m-vertices which must be in block P_1. Then an AND-OR realization of the output function y_1 must have at least H(m,n)+1 gates since there must be one AND gate for every m-vertex in P_1 as a consequence of Lemma 3.7.

Similarly, an OR-AND realization of y_2 must have at least H(m,n)+1 gates since one OR gate corresponds to each m-vertex not in P_2. Any other realization of y_1 or y_2 under this partition can only require more gates. The same is true if a partition is used where $|\text{m-vertices in } P_1| > H(m,n)$ or $|\text{m-vertices in } P_2| > H(m,n)$. Therefore, a minimal two level realization of an m-out-of-n checker must have at least $2 \cdot H(m,n)+2$ gates.

This minimum can be achieved by following the procedure just discussed. First, the m-vertices should be partitioned according to Corollary 3.8 such that the number of m-vertices in P_1 is minimum, then y_1 should be realized using an AND-OR circuit and y_2 using an OR-AND circuit. The inputs to the AND gates which realize y_1 correspond to the 1 positions in the vertices in P_1, and the inputs to the OR gates which realize y_2 correspond to the 0 positions of the vertices in P_1. It is easy to see that such a realization performs the mapping specified by P_1 and P_2 and must be a totally self-checking checker.

Example 3.4: Construction of a minimal two-level 3-out-of-6 TSC checker.

Let $P_1 = \{(1,1,1,0,0,0),(1,1,0,1,0,0),(0,0,1,1,1,0),$
$(0,0,1,1,0,1),(1,0,0,0,1,1),(0,1,0,0,1,1)\}$;

$P_2 = \{\text{all 3-vertices not in } P_1\}$.

This partition satisfies Corollary 3.8 and $|P_1|$ is minimum. The checker appears in Figure 3.5. □

A nonexhaustive procedure for generating a partition such that $|\text{m-vertices in } P_1| = H(m,n)$ has not been found, and finding such a partition is believed to be quite difficult in the general case. In fact, the numbers $H(m,n)$ are not known in general, although we can derive a lower bound on $H(m,n)$ which appears to be rather tight for $m \simeq \lfloor \frac{n}{2} \rfloor$.

Let $H^-(m,n)$ be the minimum number of m-vertices of the n-cube which are required to cover all (m-1)-vertices, and let $H^+(m,n)$ be the minimum number of m-vertices necessary in order for all (m+1)-vertices to cover at least one m-vertex. We observe that $H(m,n) \geq \max(H^-(m,n),H^+(m,n))$.

The covering of (m-1)-tuples by m-tuples has been treated rather extensively in combinatorics [30,31]. From [30] we get a bound on $H^-(m,n)$;

$$H^-(m,n) \geq \left\lceil \frac{n}{m} \left\lceil \frac{n-1}{m-1} \left\lceil \cdots \left\lceil \frac{n-m+3}{3} \left\lceil \frac{n-m+2}{2} \right\rceil \right\rceil \cdots \right\rceil \right\rceil \right\rceil.$$

Due to the same type of 'duality' observed between Conditions 1) and 2) of Corollary 3.8,

$$H^+(m,n) = H^-(n-m,n).$$

Therefore,

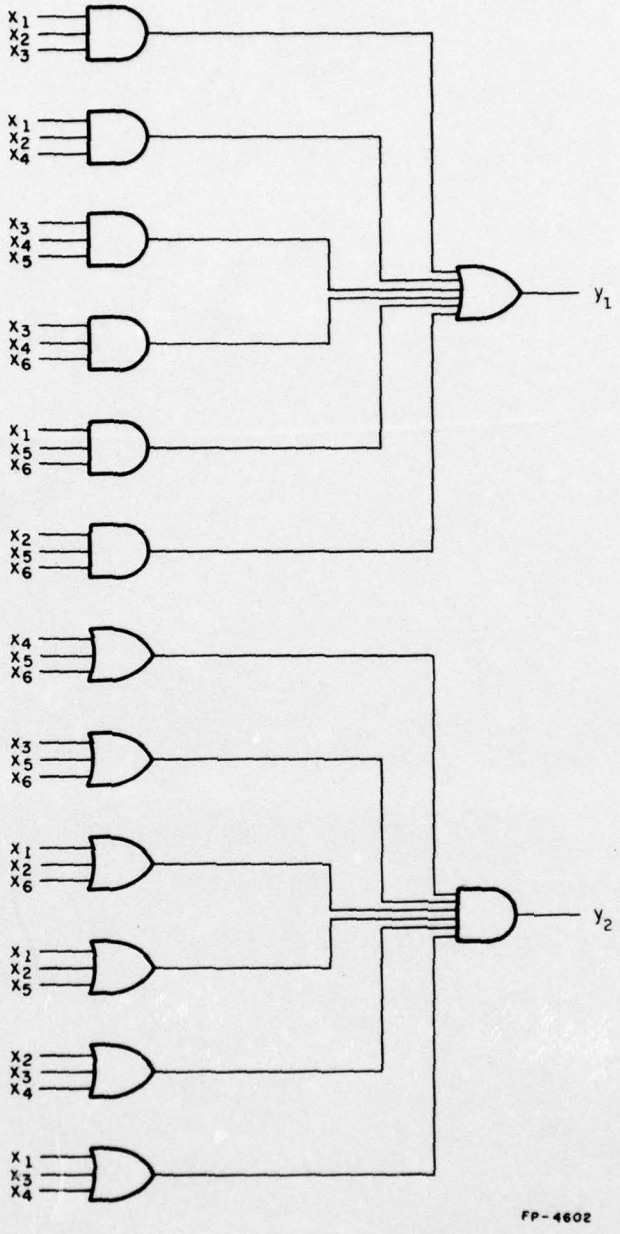


Figure 3.5. A minimal two-level 3-out-of-6 TSC checker.

$$H^{+}(m,n) \geq \left\lceil \frac{n}{n-m} \left\lceil \frac{n-1}{n-m-1} \left\lceil \cdots \left\lceil \frac{m+3}{3} \left\lceil \frac{m+2}{2} \right\rceil \right\rceil \cdots \right\rceil \right\rceil \right\rceil,$$

and

$$H(m,n) \geq \max \left\lceil \frac{n}{m} \left\lceil \frac{n-1}{m-1} \left\lceil \cdots \left\lceil \frac{n-m+3}{3} \left\lceil \frac{n-m+2}{2} \right\rceil \right\rceil \cdots \right\rceil \right\rceil \right\rceil,$$

$$\left\lceil \frac{n}{n-m} \left\lceil \frac{n-1}{n-m-1} \left\lceil \cdots \left\lceil \frac{n+3}{3} \left\lceil \frac{m+2}{2} \right\rceil \right\rceil \cdots \right\rceil \right\rceil \right\rceil.$$

For H(2,4), H(3,6), and H(4,8), this bound is 2, 6, and 14 respectively. Actual partitions have been found which meet these bounds exactly. Thus, for H(k,2k) it appears that the bound is very close to being exact. We further observe that for k-out-of-2k codes we can expect H(k,2k) to be considerably less than $\binom{2k}{2}/2$ which would result from a partition with an equal number of code vertices in P_1 and P_2.

For H(2,5) the bound is 4; however, it can be shown that H(2,5) = 5. This is evidence that as n moves further away from $\lfloor \frac{n}{2} \rfloor$ the bound becomes less exact, and other examples confirm this trend.

For many m and n, the conditions of Corollary 3.8 cannot be met according to Theorem 3.10. In these cases, a serial interconnection is necessary in order to construct a TSC checker. Figure 3.6 shows such an interconnection containing two blocks, G_1 and G_2.

The block G_2 is a k-out-of-2k TSC checker with a set of code words C which are sufficient to test it. G_1 maps the members of an m-out-of-n code where $\binom{2k}{k} \geq \binom{n}{m} \geq |C|$ one-to-one onto C and any $\binom{n}{m} - |C|$ other k-out-of-2k code words. Then the interconnection must be TSC and u-error preserving according to Theorem 3.7.

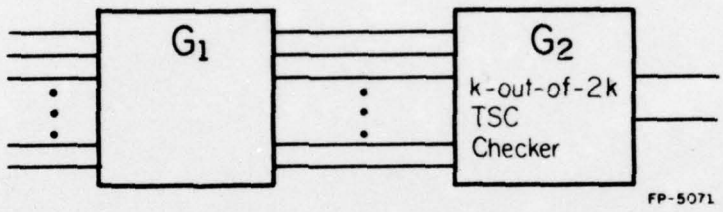


Figure 3.6. A TSC checker formed as an interconnection of two blocks.

In [32], Anderson and Metze presented designs for two-output k-out-of-2k TSC checkers which require 2^k code inputs to sufficiently exercise them. It is interesting to note that the partition of code words used in these checkers satisfies Corollary 3.8 as is shown in [14]. Since $\binom{2k}{k} \geq 2^{k+1}$ for $k \geq 3$, the k-out-of-2k checker can be used as G_2 when the m-out-of-n code input to G_1 has between 2^k and 2^{k+1} members. Thus, for any m-out-of-n code with more than 8 members, one of the checkers in [32] will suffice for G_2. For codes with fewer than 8 members, a similar decomposition can be used except for 1-out-of-3 (2-out-of-3) and 1-out-of-7 (6-out-of-7) codes.

In [15] Reddy presented two-output TSC checkers for k-out-of-2k codes which require only 2k code inputs as tests making it possible to construct a 1-out-of-7 and a 6-out-of-7 two-output checker. By Theorem 3.8, a two-output checker for the 1-out-of-3 and 2-out-of-3 codes which is TSC with respect to unidirectional faults does not exist.

This method of constructing TSC checkers for an arbitrary m-out-of-n code leads to a theorem dealing with checkers for the more general class of closed unordered codes.

Theorem 3.12: A two-output checker which is TSC with respect to unidirectional faults exists for all closed unordered codes except the 1-out-of-3 and 2-out-of-3 codes.

Proof: Any closed unordered code can be mapped one-to-one onto a set of code words sufficient to check a two output k-out-of-2k checker (except for the 1-out-of-3 and 2-out-of-3 codes). Such a mapping can be realized so that it is either u-error preserving (Corollary 3.4) or code

disjoint (Corollary 3.5). Thus, the serial interconnection of this block G_1 and the appropriate k-out-of-2k checker G_2 results in a two output TSC checker which is either u-error preserving or code disjoint. Q.E.D.

We now present a method of constructing two-output k-out-of-2k TSC checkers which have a regular array structure, which require only 2k tests, and which have fewer gates and approximately half the levels of Reddy's checkers. We do not include a proof that they operate as claimed since this would be rather lengthy. Nevertheless, one can convince oneself that they operate correctly by studying several particular cases.

Figure 3.7 illustrates the construction of our k-out-of-2k checkers. The 2k words which are sufficient to test the checkers are the same as those in [15], that is, all 2k circular shifts of k 1's followed by k 0's.

Table 3.1 summarizes the number of levels, gates, and tests required for the k-out-of-2k checker designs which appear here and in the literature.

	no. levels	no. gates	no. tests
Anderson and Metze [32]	≥ 3	not specified	2^k
Reddy [15]	2k-1	$2k^2-k+1$	2k
This thesis	k	$2k^2-2k+2$	2k
Reddy [15]	2	$\binom{2k}{k}+2$	$\binom{2k}{k}$
This thesis	2	$2H(k,2k)^*+2$	$2H(k,2k)^*$

*H(3,6) = 6
H(4,8) = 14
$H(5,10) \simeq 50$
otherwise H(k,2k) is not known.

Table 3.1. Summary of TSC checkers.

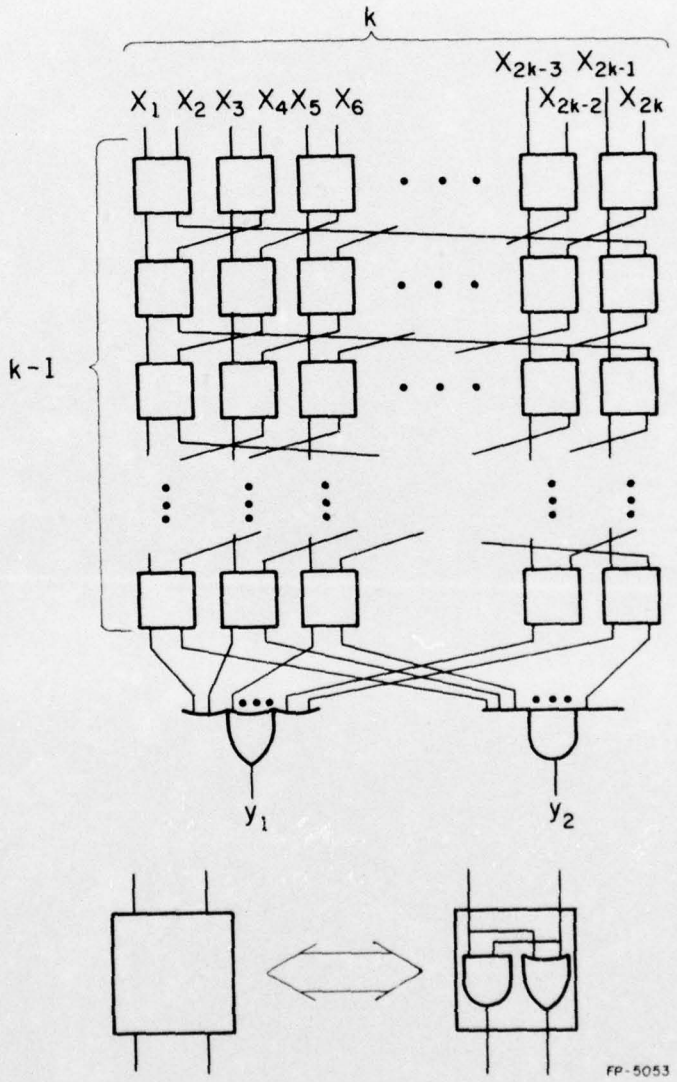


Figure 3.7. A k-out-of-2k TSC checker requiring 2k tests.

3.4.3. TSC Circuits Using Systematic Unordered Codes

A systematic code is one in which the information bits and the redundant check bits can be separately identified. In an application where a TSC circuit is desired, but where some specific set of (possibly ordered) information bits are required, it may be necessary to add some check bits to obtain a code that is unordered.

We give a general method for taking an arbitrary set of equal length information words and determining check bits for each word which yield an unordered code. Since no two information words are the same, we can replace the covering relation $\leq$ which we have defined with the relation $<$ which is a partial ordering [22]. To make an arbitrary set of information vertices unordered:

1) Topologically sort [33] the vertices into a list. That is if $x_i < x_j$ then x_i precedes x_j in the list.
2) Break the sorted list into antichains which contain only contiguous members of the list.
3) If m antichains are used in step 2), take any m different equal length vertices and topologically sort them according to the partial ordering $>$; i.e. if $y_i > y_j$ then y_i precedes y_j in the list.
4) Assign the ith vertex in the sorted check list to all the members of the ith antichain in the sorted information list.

That the above procedure must produce an unordered code is rather obvious since for any member of the code z_i to cover another member $z_i > z_j$, $x_i > x_j$ and $y_i \geq y_j$. If $y_i = y_j$ then x_i and x_j must be in the same antichain so $x_i \not> x_j$. If $y_i > y_j$ then y_i precedes y_j in the check list, but $x_i > x_j$

implies x_j precedes x_i in the information list. Therefore, it is impossible that y_i be assigned to x_i and y_j to x_j.

Example 3.5: Given the information vertices (0,0,0),(0,0,1), (0,1,0),(0,1,1),(1,0,0),(1,0,1),(1,1,0),(1,1,1), construct a systematic unordered code.

After steps 1 and 2 of the above procedure, we have:

(000)
(010)
(001)
(100)
(101)
(110)
(011)
(111) .

4 antichains

Since there are 4 antichains, a set of check vertices is:

(11)
(01)
(10)
(00).

Then, we arrive at the unordered code:

(00011)
(01001)
(00101)
(10001)
(10110)
(11010)
(01110)
(11100).

We could also have used 8 antichains and 8 check words to arrive at the code:

(000 111)
(010 101)
(001 110)
(100 011)
(101 010)
(110 001)
(011 100)
(111 000). □

Definition 3.5: A systematic unordered code is optimal if a minimum number of check bits is used to make the code unordered.

Lemma 3.8: If m is the length of the longest chain in the information vertices then at least m different check words are necessary to make the code unordered.

Proof: If any two members of any chain have the same check word then the code cannot be unordered. Q.E.D.

A well known theorem from combinatorics, Dilworth's Theorem [34], is applicable at this point. The version presented here is actually a dual form of Dilworth's Theorem which we have further modified to consider the topological sort of the information words. The proof given here is derived from the one in [26].

Lemma 3.9: (Dilworth) Let P be a partially ordered set whose longest chain is length m. Then, there exists a topological sorting of P such that the members of the sorted list can be broken into m disjoint antichains each containing contiguous members of the list.

Proof: (by induction on m) When m = 1, the theorem holds trivially. Assume that the theorem holds for m-1. Let P be a partially ordered set whose longest chain is length m. Let M denote the set of all minimal elements of P. Clearly, M is nonempty. Place the members of M at the

beginning of the sorted list. Since M contains minimal elements, no member of M can cover any member of P-M. Furthermore, the members of M must form an antichain. The maximal chain length in P-M is m-1 since all the minimal elements have been removed. By the induction hypothesis there is a topological sorting of P-M which can be placed after M in the list to get a sorting with M contiguous antichains. Q.E.D.

Theorem 3.13: Given a set of information vertices whose longest chain is length m, then $\lceil \log_2(m) \rceil$ check bits are necessary and sufficient to form an unordered code.

Proof: (Necessity): From Lemma 3.8, at least m different check words are needed which would require at least $\lceil \log_2(m) \rceil$ bits.

(Sufficiency): The information vertices can be sorted according to the procedure outlined in the proof of Lemma 3.9 to get m antichains. This means that m different check words are sufficient, and $\lceil \log_2(m) \rceil$ bits are sufficient to form m different check words. Q.E.D.

We have just given a procedure for generating check bits to make any set of information words unordered. Thus, if a TSC functional block is desired which uses a systematic code, we only need to follow the above procedure and to use an inverter-free realization followed by reduction with respect to undetected faults.

Furthermore, if an optimal systematic code is desired, the method outlined in Lemma 3.9 can be used.

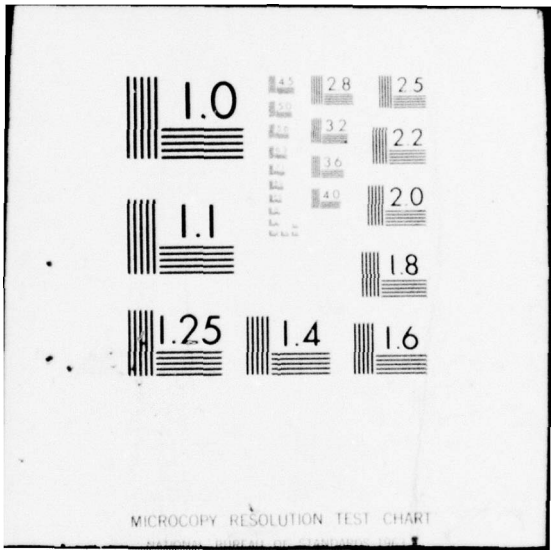
We define a systematic code to be complete if it has k information bits and 2^k different code words. From Theorem 3.13, any optimal complete systematic (OCS) unordered code must have $\lceil \log_2(k+1) \rceil$ check bits. Berger

AD-A031 430 ILLINOIS UNIV AT URBANA-CHAMPAIGN COORDINATED SCIENCE LAB F/G 9/2
THE DESIGN OF TOTALLY SELF-CHECKING COMBINATIONAL CIRCUITS.(U)
AUG 76 J E SMITH DAAB07-72-C-0259
UNCLASSIFIED R-737 NL

2 OF 2
AD
A031430

END

DATE
FILMED
11-76


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART

codes [35] are the best known of the OCS codes. In Berger codes, the check bits form a binary count of the number of 0's in the information bits. Berger codes can be constructed by the procedure given earlier, and the first code given in Example 3.5 is in fact a Berger code.

While the Berger codes are OCS codes they are not all closed, and, therefore, do not satisfy any of the theorems given for the construction of u-error preserving functional blocks. In fact, the only Berger codes which are closed are those which have k information bits and use all combinations of the $\lceil \log_2(k+1) \rceil$ check bits. In particular those for which $k = 2^p - 1$ for $p = 1,2,\ldots$.

Other closed OCS codes can be found; a class where $k = 2^p$, $p = 1,2,\ldots$ presented by Ashjaee and Reddy [36] will be discussed in the next section. However, closed OCS codes for all k have not been found. At present, the only OCS codes for which two output checkers have been found are closed. These checkers can be constructed as a result of Theorem 3.12. The checkers given in [36] have this serial interconnection structure. In some cases, it may be possible to partition a closed OCS code in such a way that the conditions of Theorem 3.6 are satisfied, so that the functional block realizing the initial mapping in the interconnection need not be one-to-one.

Example 3.6: A TSC checker for the Berger code with 3 information bits is shown in Figure 3.8. In this case, a 4-block partition of the code words can be found which satisfies Theorem 3.6. □

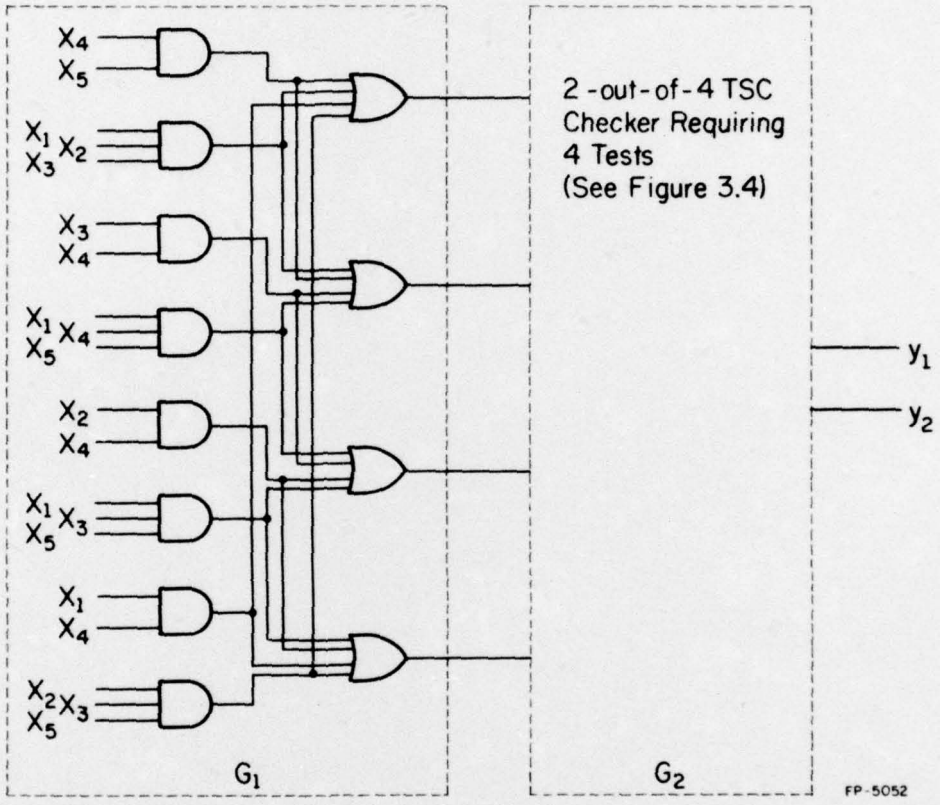


Figure 3.8. A TSC checker for the Berger code with 3 information bits.

3.4.4. TSC Circuits Using Concatenated Codes

We begin by giving two definitions.

Definition 3.6: The concatenation of a vertex t with m coordinates and a vertex u with n coordinates is a vertex v with m+n coordinates whose first m coordinates are identical to those in t, and whose last n coordinates are identical to the coordinates in u.

For example, the concatenation of (0,1,0,1) and (1,1) is (0,1,0,1,1,1).

Definition 3.7: The concatenation of two codes T and U of size $|T|$ and $|U|$ respectively is a code V of size $|T| \cdot |U|$ which is formed by concatenation each vertex in T with every vertex in V.

Example 3.7: The concatenation of the codes $T = \{(0,1),(1,0)\}$ and $U = \{(0,0),(1,1)\}$ yields the concatenated code $V = \{(0,1,0,0),(0,1,1,1),(1,0,0,0),(1,0,1,1)\}$. □

The use of concatenated codes allows us a much larger set of unordered codes than the ones previously given. For example, the two-rail codes where each bit and its complement appear in a code word is an extreme example of concatenated codes.

Obviously, the concatenation of any two unordered codes is an unordered code. Furthermore, the following theorem points out that the same is true for closed unordered codes.

Theorem 3.14: The concatenation of two closed unordered codes is a closed unordered code.

Proof: Let $t \cdot u$ represent the concatenation of two code words t and u, one from each of the codes being concatenated. Since both codes are

closed, changing any bit in t will yield some word t', which is ordered with respect to some other code word, say t". Obviously t'·u is ordered with respect to t"·u; so in the concatenated code, changing any bit in the t part yields a word which is ordered with respect to some other code word. By a similar argument, it can be shown that changing any bit in the u part yields a word which is ordered with respect to some other code word. Therefore, the same is true if <u>any</u> bit is changed, so the concatenated code must be closed. Q.E.D.

Example 3.8: Construction of a complete OCS code where the number of information bits $k = 2^p$, $p = 1,2,...$

The construction of complete OCS codes where $k = 2^p-1$ was given in the previous section. An OCS code where $k = 2^p-1$ requires p check bits, while for $k = 2^p$, p+1 check bits are needed. If the code $\{(0,1),(1,0)\}$ is concatenated to the codes where $k = 2^p-1$, one additional information bit is added, and one redundant check bit is added. This gives a code with 2^p information bits and the minimum p+1 check bits. From Theorem 3.14, since the codes with $k = 2^p-1$ developed earlier are closed and the code $\{(0,1),(1,0)\}$ is obviously closed, the resulting code must also be closed. □

The codes described in Example 3.8 have been previously given in [36], where they are called "modified Berger codes."

Theorem 3.6 shows that if the partition of a code meets certain conditions, then any inverter-free realization will be u-error preserving. It has further been shown that a satisfactory two block partition is very important when a TSC checker is being designed. We now show how to

construct a satisfactory partition for a closed concatenation code given satisfactory partitions of the component closed codes. Say the closed codes T and U are concatenated to form code V. Assume we are given an n-block partition of T and an n-block partition of U each of which satisfies Theorem 3.6. Name the partition blocks $t_1, t_2, \ldots, t_n$ and $u_1, u_2, \ldots, u_n$. Then a satisfactory n-block partition of V can be formed as follows:

block 1	block 2	block 3	...	block n-1	block n
$t_1 \cdot u_1$	$t_1 \cdot u_2$	$t_1 \cdot u_3$	...	$t_1 \cdot u_{n-1}$	$t_1 \cdot u_n$
$t_2 \cdot u_2$	$t_2 \cdot u_3$	$t_2 \cdot u_4$	...	$t_2 \cdot u_n$	$t_2 \cdot u_1$
$t_3 \cdot u_3$	$t_3 \cdot u_4$	$t_3 \cdot u_5$	...	$t_3 \cdot u_1$	$t_3 \cdot u_2$
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
$t_n \cdot u_n$	$t_n \cdot u_1$	$t_n \cdot u_2$		$t_n \cdot u_{n-2}$	$t_n \cdot u_{n-1}$

That is, block v_i of the satisfactory partition of V has as members $\bigcup_{j=1}^{n} t_j \cdot u_{(j+i-1) \bmod n}$.

Theorem 3.15: The partition of the code V as described above satisfies Theorem 3.6.

Proof: Assume some bit in the t part of some code word in block v_i is changed. Say the code word which was changed was in the subblock $t_\ell \cdot u_{(\ell+i-1) \bmod n}$. Then, there must be some other block in the partition of T, say t_m, such that the modified word in t_ℓ is ordered with respect to some word contained in it. Then, the modified word in v_i must be ordered with respect to some word in $t_m \cdot u_{(\ell+i-1) \bmod n}$. However, $t_m \cdot u_{(\ell+i-1) \bmod n} \notin v_i$. Therefore, the code word which is changed is ordered with respect to some code word in a different block.

A similar argument can be used if some bit in the u part of some code word in v_i is changed. Q.E.D.

It is also true that any partition of a concatenated code which satisfies Theorem 3.6 must be decomposable into the form given above (allowing permutations, of course).

A very important unordered code is the two-rail code which is one of the simplest concatenated codes. In a two-rail code, each bit of information is represented by both the true and complemented values of the bit. For example, the information (0,1,0) is encoded as (0,1,1,0,0,1).

All of the two-rail codes can be constructed by repeated concatenation of the code $\{(0,1),(1,0)\}$. Since this simple code is obviously closed, by repeated application of Theorem 3.14, any two-rail code is closed.

The partition $\{(0,1)\}|\{(1,0)\}$ of the initial 1-out-of-2 code satisfies Corollary 3.7. Therefore, repeated application of the method given above can be used to generate two-block partitions which satisfy Theorem 3.6 for all the two-rail codes.

Example 3.9: Construction of a TSC checker for the two-rail code with 3 two-rail bits (6 total bits).

Initially, we can concatenate the code $\{(0,1),(1,0)\}$ with itself, and using the above method, we get the satisfactory two block partition:

block 1	block 2
(0,1,0,1)	(0,1,1,0)
(1,0,1,0)	(1,0,0,1).

Then $\{(0,1),(1,0)\}$ can be concatenated to this code, and the method can be used again to get the satisfactory partition:

block 1	block 2
(0,1,0,1,0,1)	(0,1,0,1,1,0)
(1,0,1,0,0,1)	(1,0,1,0,1,0)
(0,1,1,0,1,0)	(0,1,1,0,0,1)
(1,0,0,1,1,0)	(1,0,0,1,0,1).

Then, by Theorem 3.6, any inverter-free realization of a mapping of block 1 onto (1,0) and block 2 onto (0,1) will be a P-FS and can be easily made into a TSC checker. □

A serial interconnection of u-error preserving blocks can be used to realize most completely specified single-output functions in a u-error preserving manner with two-rail codes at both the input and output. The only functions which cannot be realized in this way are those which violate Theorem 3.8. Furthermore, at most four logic levels are necessary for any realizable function.

We now present a method for constructing u-error preserving two-rail functional blocks which realize any realizable single-output function.

If we are given a completely specified mapping, let A be those two-rail code words which map to (1,0), and let B be those which map to (0,1). $|A|$ and $|B| \geq 2$ if the function is realizable by Theorem 3.8. Let $C|D$ represent the partition of the two-rail code (there is only 1) which satisfies Theorem 3.6. If $A = C$ or $B = C$, then the given partition is satisfactory and any inverter-free realization gives the desired result. Otherwise, partition A and B into $A'|A''$ and $B'|B''$ such that $A' \subset C$ and $A'' \subset D$ and $B' \subset C$ and $B'' \subset D$. If A' is empty, place any member of A'' into it, if A'' is

empty, place any member of A' into it, and similarly for B' and B".

We now have the four block partition A'|A"|B'|B" where each block is nonempty. If A' is mapped onto (0,1,0,1), A" onto (1,0,1,0), B' onto (0,1,1,0) and B" onto (1,0,0,1), then any realization is u-error preserving by Theorem 3.6.

Now if (0,1,0,1) and (1,0,1,0) are mapped onto (1,0), and (0,1,1,0) and (1,0,0,1) are mapped onto (0,1) then any inverter-free realization will be u-error preserving by Theorem 3.6, and the interconnection of the two blocks will give the desired mapping. Furthermore, if two-level realizations are used for both blocks of the interconnections, a total of only four levels is required.

Example 3.10: The realization of a u-error preserving two-rail functional block for the mapping given below.

input x	$G(x,\phi)$
(0,1,0,1,0,1)	(0,1)
(0,1,0,1,1,0)	(0,1)
(0,1,1,0,0,1)	(0,1)
(0,1,1,0,1,0)	(1,0)
(1,0,0,1,0,1)	(1,0)
(1,0,0,1,1,0)	(1,0)
(1,0,1,0,0,1)	(1,0)
(1,0,1,0,1,0)	(1,0)

Then $A = \{(0,1,1,0,1,0),(1,0,0,1,0,1),(1,0,0,1,1,0),(1,0,1,0,0,1),(1,0,1,0,1,0)\}$

and $B = \{(0,1,0,1,0,1),(0,1,0,1,1,0),(0,1,1,0,0,1)\}$.

Now, in this case, $C = \{(0,1,0,1,0,1),(1,0,1,0,0,1),(1,0,1,0,0,1),(0,1,1,0,1,0),(1,0,0,1,1,0)\}$ and $D = \{(0,1,0,1,1,0),(1,0,1,0,1,0),(0,1,1,0,0,1),(1,0,0,1,0,1)\}$ (from Example 3.11). So,

$$A' = \{(0,1,1,0,1,0),(1,0,0,1,1,0),(1,0,1,0,0,1)\}$$

$$A'' = \{(1,0,0,1,0,1),(1,0,1,0,1,0)\}$$

$$B' = \{(0,1,0,1,0,1)\}$$

$$B'' = \{(0,1,0,1,1,0),(0,1,1,0,0,1)\}.$$

Now if we map $A' \rightarrow (0,1,0,1)$

$A'' \rightarrow (1,0,1,0)$

$B' \rightarrow (0,1,1,0)$

$B'' \rightarrow (1,0,0,1)$

using block G_1 and then use a block G_2 to map

$(0,1,0,1) \rightarrow (1,0)$

$(1,0,1,0) \rightarrow (1,0)$

$(0,1,1,0) \rightarrow (0,1)$

$(1,0,0,1) \rightarrow (0,1)$,

the resulting interconnection shown in Figure 3.9 is a u-error preserving TSC realization of the desired function. □

As another example of the techniques presented in this chapter, we will consider two-rail full adders, which have two-rail inputs and generate a two-rail sum and carry signal.

Example 3.11: Realization of a u-error preserving TSC two-rail full adder. Such circuits realize the following mapping:

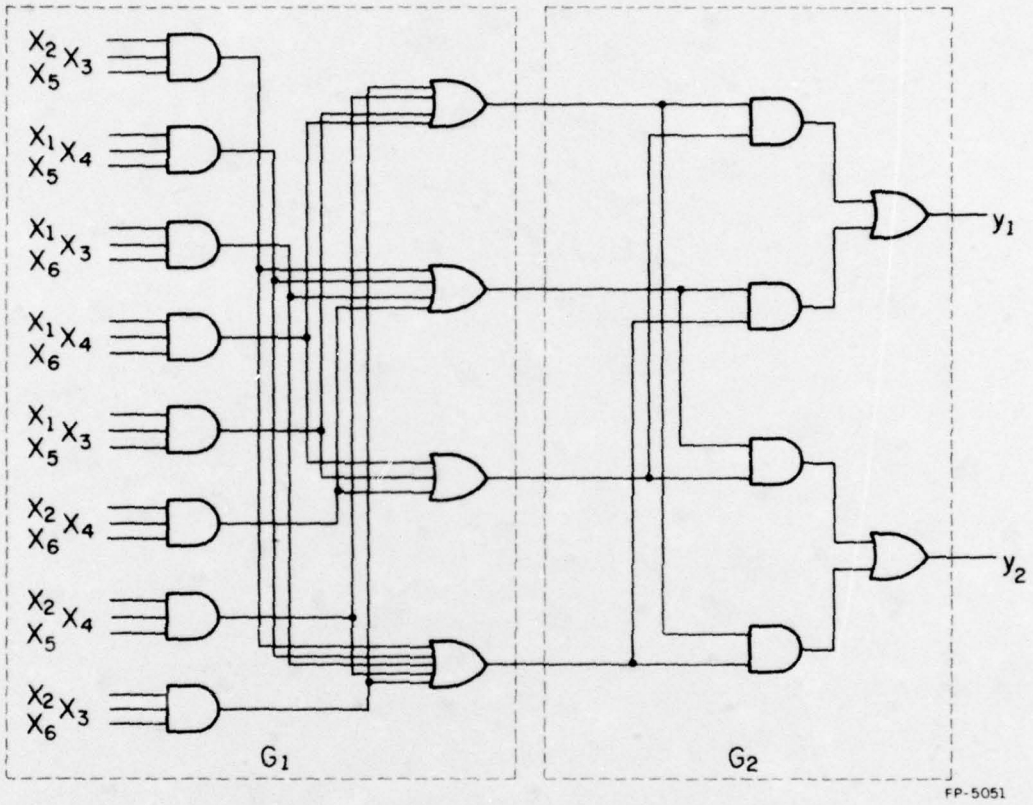


Figure 3.9. A u-error preserving TSC realization of the function given in Example 3.10.

a_1	a_2	c_i	s	c_o
(0,1,	0,1,	0,1)	(0,1,	0,1)
(0,1,	0,1,	1,0)	(1,0,	0,1)
(0,1,	1,0,	0,1)	(1,0,	0,1)
(0,1,	1,0,	1,0)	(0,1,	1,0)
(1,0,	0,1,	0,1)	(1,0,	0,1)
(1,0,	0,1,	1,0)	(0,1,	1,0)
(1,0,	1,0,	0,1)	(0,1,	1,0)
(1,0,	1,0,	1,0)	(1,0,	1,0)

Such a mapping immediately satisfies Theorem 3.6. Therefore, any inverter-free two-rail adder is u-error preserving and can be made TSC simply by reduction with respect to any undetected faults. □

Perhaps one of the most important features of concatenated codes is that they allow efficient closed unordered codes of a large number of sizes. That is, if we were restricted to m-out-of-n codes, for some code sizes the only closed code would be inefficient. For example, the only m-out-of-n code with 100 members is the 1-out-of-100 code which is extremely inefficient. However, the concatenation of two 2-out-of-5 codes is a closed code with 100 members which is very efficient since it only requires 10 bits.

4. TSC CIRCUITS FOR SINGLE FAULTS

4.1. Introduction

Probably the most commonly accepted fault assumption for TSC circuits is the single fault assumption. While the unidirectional fault assumption discussed in the previous chapter is useful for some applications, such as memories, in today's logic circuitry the single fault assumption seems to be sufficient in the majority of cases. In addition, single faults make up a smaller fault set than unidirectional faults, so they force fewer restrictions on circuit structure (e.g. inverter-freeness).

In this chapter we give two basic circuit structures which yield TSC circuits for single faults. Both are structures which are path-fault secure. The first is found by allowing conversion of the circuits described in Chapter 3 to ones which contain inverters, and the second uses structures which are formed by a technique known as "slicing." We also discuss interconnections for each type of structure, and the interconnection of one structure type to the other.

4.2. Substitution Theorems

We begin by presenting a set of theorems dealing with the substitution of different realizations of a subnetwork in a TSC functional block.

Consider a functional block G with input code space A and output code space B. Embedded in G is a single-output subnetwork N with the inputs $i_1, i_2, \ldots, i_\ell$ and output n as shown in Figure 4.1.

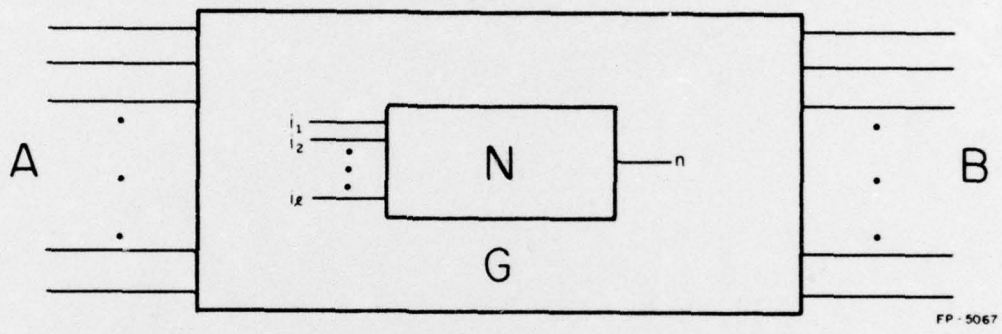


Figure 4.1. The functional block G with subnetwork N.

Theorem 4.1: If the functional block G is fault secure with respect to single faults and the subnetwork N is replaced by any subnetwork N' which realizes the same function that N does, then the new functional block G' is fault secure with respect to single faults.

Proof: Let f be a single stuck-at fault occurring in N'. Then the line n must be on any sensitized path from f to any output. If f is sensitized to some output under input a, then the line n must be on the sensitized path and must have an incorrect value, say $\bar{d}$ instead of a correct d. Since line n s-a-d is a fault for which G is fault secure, $G(a,\{n/d\}) = G(a,\phi)$ or $G(a,\{n/d\}) \notin B$. Since N and N' are single-output subnetworks, $G(a,\{n/d\}) = G'(a,\{n/d\})$. Furthermore, any path from f to an output must pass through n, so $G'(a,f) = G'(a,\{n/d\})$. Therefore, $G'(a,f) = G'(a,\phi)$ or $G'(a,f) \notin B$.

If f is a single stuck-at fault occurring in G' but not in N', the fact that the function realized by N' is the same as that realized by N means that sensitization of f to any output under any input will be unaffected. So if G is fault secure with respect to f, G' will be fault secure with respect to f. Q.E.D.

Corollary 4.1: If G is TSC with respect to single faults and the subnetwork N is replaced by any subnetwork N' which realizes the same function as N does, and each single fault in N' can be sensitized to at least one output of G by a member of A, then G' is TSC with respect to single faults.

Proof: From Theorem 4.1, G' must be fault secure with respect to single faults. Since a single fault f can be sensitized to an output by a member of A, $\exists a \in A$ such that $G'(a,f) \neq G'(a,\phi)$. The substitution of N'

cannot affect the testing of any f not in N', so by Definition 2.3', G' is TSC with respect to single faults. Q.E.D.

If the subnetwork N' is not completely tested by members of A and it is substituted into G, then a nontested fault f may convert N' into a 'new' network N" such that under a second fault f", f' and f" conspire in such a way that an incorrect code word appears at the output. This can happen if the value of n under f is such that it causes additional paths to be sensitized from f' to the output of G'.

If G is P-FS, then this cannot happen as we will now show.

Theorem 4.2: If G is P-FS with respect to single faults and the subnetwork N is replaced by any subnetwork N' which realizes the same function as N does, then the new network G' is P-FS with respect to single faults.

Proof: From the definition of P-FS, any single fault in N' must have a sensitization set which is a subset of $\sigma(\{n/1\}) \cup \sigma(\{n/0\})$. This is also true for any fault in N. Therefore, no new sensitization vectors are added to the sensitization set for the set of single faults by the substitution. Q.E.D.

The preceding theorems give methods for modifying circuits which are known to be TSC or E-TSC with respect to single faults without changing the TSC or E-TSC property. Since none of the substitutions change the function realized by G, if G is error preserving with respect to some error set E then G' must also be error preserving with respect to E.

4.3. Conversion of Inverter-Free Networks

Since the circuits which are designed according to the methods given in the previous chapter are TSC (E-TSC) with respect to unidirectional faults, they must also be TSC (E-TSC) with respect to single faults. This is because the single fault set is a subset of the unidirectional fault set. Since they are TSC with respect to single faults, the substitution theorems outlined in the previous section apply and can be used to convert them to circuits with inverters while retaining the TSC (or E-TSC) property.

For example, in some applications, it may be desirable to convert a network to an all NAND or all NOR network. We can easily perform such a conversion on our TSC (E-TSC) inverter-free circuits by using Corollary 4.1 and Theorem 4.2. Hayes' normal NAND conversion [37] is one such conversion which satisfies Corollary 4.1 and Theorem 4.2 at each step. The following method which is a simplified version of Hayes' will also work.

1) Replace each AND gate with a NAND gate followed by an inverter (a one-input NAND gate). Replace each OR gate with a NAND gate which has an inverter on each of its inputs.
2) Convert two consecutive inverters on a line to a simple line.

Example 4.1: Construction of an all NAND 2-out-of-4 TSC checker for single faults beginning in Figure 4.2 with an inverter-free 2-out-of-4 TSC checker for unidirectional faults. □

Circuits which are converted to circuits with inverters as just described are not only TSC with respect to single faults, but if a network G is error preserving with respect to an error set E, then the new all NAND network is also error preserving with respect to E since it realizes the

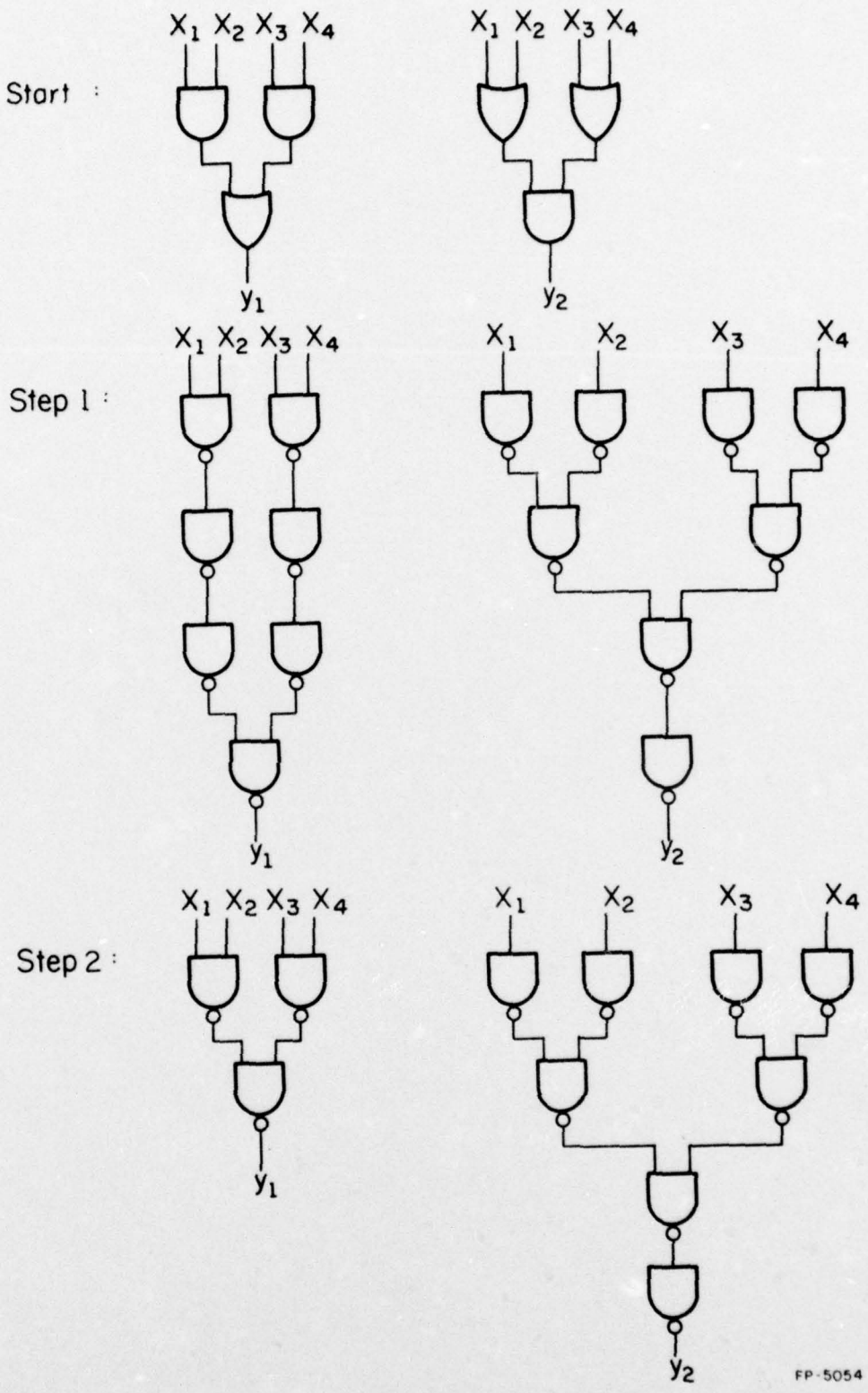


Figure 4.2. Construction of an all NAND TSC checker.

same function as G. Also, the set of output noncode words which can result from single faults in the NAND network is a subset of those which can be produced by unidirectional faults in G. Therefore, any interconnection of blocks which is TSC (E-TSC) with respect to unidirectional faults as constructed in Chapter 3 is TSC (E-TSC) with respect to single faults when it is converted to an all NAND network. In addition, any checker output is still a checker output in the new interconnection.

The reduction of the fault set to single faults also makes it possible to adapt the TSC blocks designed in Chapter 3 to TSC blocks which use less restricted input and output encodings.

Definition 4.1: An inverter-only (I-O) code translator is one which contains only inverters, and any path from an input to an output contains at most one inverter.

It is easy to see that an I-O code translator must perform a one-to-one mapping since each output bit is either always the same as some input bit or always the complement of some input bit. Furthermore, the restriction to one inverter on any path does not reduce the number of mappings which would be possible if any number of inverters were allowed. In fact, a one-level translator is more economical than a multiple-level translator.

Theorem 4.3: Let G_2 be a functional block with input code space A and output code space B which can be derived from an inverter-free network G_2' which is TSC with respect to unidirectional faults by repeated applications (possibly none)of Corollary 4.1. Let G_1 be an I-O code translator with input code space A' and output code space A, and let G_3 be an I-O code translator with input code space B and output code space B'. Then the serial

interconnection of G_1, G_2, and G_3 (Figure 4.3) forms a functional block with input code space A' and output code space B' which is TSC with respect to single faults.

Proof: Since the fault set is all single stuck-at faults on gate inputs and gate outputs, the only potential fault sites in G_1 are on inverter inputs and outputs. However, any stuck-at-d fault on the input of an inverter can be modeled as a stuck-at-$\bar{d}$ fault on its output [42]. Any single stuck-at fault on the output of an inverter in G_1 can be modeled as a unidirectional output fault, which in turn can be modeled as a unidirectional primary input fault in G_2. Any unidirectional primary input fault in G_2' can be modeled as a unidirectional stuck-at fault. Furthermore, G_2' is TSC with respect to unidirectional primary input faults. The property of being TSC with respect to unidirectional primary input faults is obviously a property of the function a block realizes, not its structure. Since G_2 and G_2' realize the same function, G_2 must be TSC with respect to unidirectional primary input faults, also.

All of the single stuck-at faults in the serial interconnection of G_1 and G_2 can be modeled as either single stuck-at faults in G_2 or unidirectional faults at the primary inputs of G_2. Since G_2 is TSC with respect to both types of faults, the interconnection of G_1 and G_2 must be TSC with respect to single faults.

As we have observed earlier, the I-O code translators can only perform one-to-one mappings. Therefore, since each code output is mapped onto by some code input, it is impossible for a noncode input to map onto a code output; therefore, G_3 must be code disjoint.

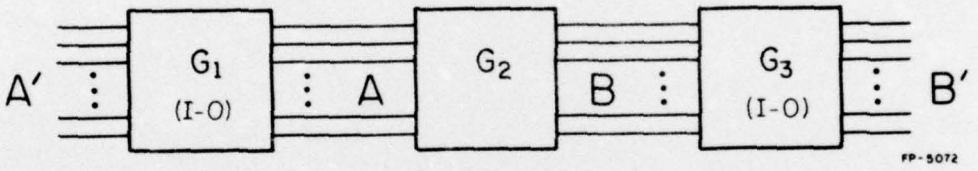


Figure 4.3. The serial interconnection of G_1, G_2, and G_3.

A single stuck-at fault in G_3 can be modeled as a unidirectional primary output fault as we observed for G_1. Say the affected outputs are $y_1, y_2, \ldots, y_n$. Then it is necessary that all of the $y_i = \overline{x}_j$ for some particular input x_j to G_3 due to the structure of the circuit and the assumed fault set. If no other input bits are related to x_j, then the fault can also be modeled as a stuck primary input line, which, in turn, can be modeled as a stuck output line in G_2. If any other input bits are related to x_j, then the circuit G_3 is obviously TSC with respect to the single fault since the affected and unaffected bits will conflict anytime the affected bits take on incorrect values.

If we connect the two block interconnection of G_1 and G_2 to G_3, the resulting interconnection must be TSC with respect to single stuck-at faults since G_3 is TSC with respect to all single stuck-at faults which cannot be modeled as single faults in G_2, G_3 is code disjoint, and the interconnection of G_1 and G_2 is TSC with respect to single faults. Q.E.D.

Example 4.2: Construction of an all NAND functional block which is TSC with respect to single faults and which performs the following mapping:

$x_1x_2x_3x_4$	y_1y_2
(1,1,1,1)	(1,1)
(0,0,0,0)	(1,1)
(1,0,0,1)	(0,0)
(1,0,1,0)	(0,0)
(0,1,0,1)	(0,0)
(0,1,1,0)	(0,0)

The network shown in Figure 4.4 will work according to Theorem 4.3. □

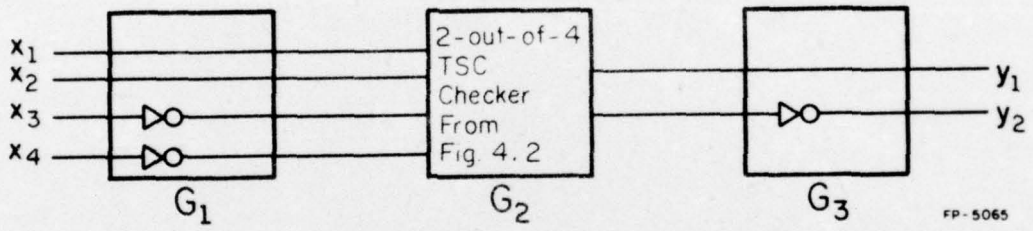

x_1
x_2
x_3
x_4
2-out-of-4
TSC
Checker
From
Fig. 4.2
y_1
y_2
G_1
G_2
G_3
FP-5065

Figure 4.4

One of the easiest codes for the construction of TSC functional blocks is the two-rail code. However, the two-rail code is the most redundant of the closed unordered codes. This code inefficiency is in conflict with a constraint imposed by today's technology--low pin counts. On the other hand, the number of logic gates used is of less importance. Theorem 4.3 allows us to effectively design functional blocks with two-rail codes while adding only two pins per functional block.

This can be done by constructing G_1 as shown in Figure 4.5, and by designing G_2 using two-rail codes. We use a trivial G_3 which consists only of lines. If a checker is connected to the outputs of G_3, and we consider G_1, G_2, G_3 and the checker to all be part of the functional block, then only the single-rail output of G_3 and the checker output are needed to give TSC operation. Such a functional block is shown in Figure 4.6.

The circuit shown in Figure 4.6 is an easily constructed TSC functional block which requires only two extra pins (the checker outputs). However, the circuit is not error preserving with respect to any error set.

4.4. Sliced Circuit Structures

We have seen that the construction of P-FS circuits is a major step to constructing TSC circuits, i.e. given a path-fault secure circuit, we only need to remove untested lines to get a TSC circuit. Perhaps the most obvious way of constructing circuits which are P-FS with respect to single faults is by using 'bit slicing' [7].

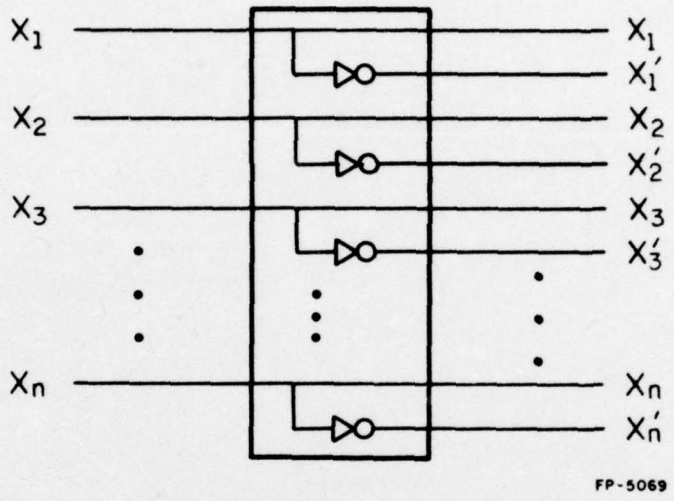


Figure 4.5. An I-O translator from single-rail to two-rail.

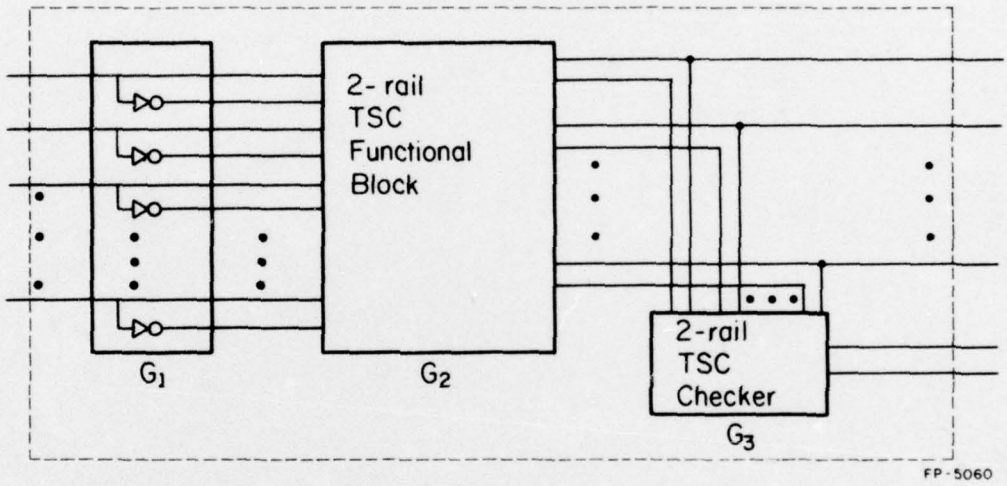


Figure 4.6. A TSC functional block with only two additional pins.

Definition 4.2: A circuit is bit sliced if each output is realized by an independent circuit having only primary inputs in common with the other output circuits.

For a bit sliced circuit the sensitization set for the single fault set contains only vectors with at most a single 1 or 0. This means that if any output code of Hamming distance two or greater is used with a bit sliced circuit, the circuit must be P-FS.

Considering bit sliced circuits leads to a generalization which we will call "byte slicing."

Definition 4.3: A circuit is b-byte sliced if the outputs can be grouped into sets of b-bits such that each set is realized by an independent circuit having only primary inputs in common with the circuits realizing the other sets.

Figure 4.7 shows a b-byte sliced circuit structure. We observe that a bit sliced circuit is 1-byte sliced.

In the following definition, two code words are at generalized Hamming distance d if the words differ in d code positions.

Definition 4.4: A b-byte distance two code is one in which the bits are grouped into bytes of size b such that if each byte is considered to be a single code position with 2^b different binary encoded output symbols, then any two members of the code are at least generalized Hamming distance 2 apart.

The simple parity code discussed in connection with bit sliced circuits is then a 1-byte distance two code.

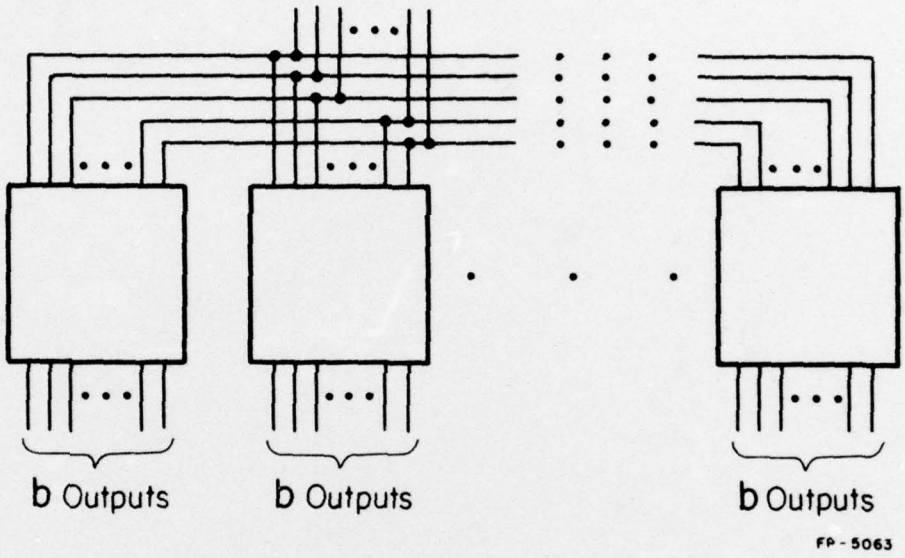


Figure 4.7. A b-byte sliced circuit.

Example 4.3: A 2-byte distance two code. Let the information bytes be in the first two positions, with a check byte in the third.

{(0,0, 0,0, 0,0),
(0,0, 0,1, 0,1),
(0,0, 1,0, 1,0),
(0,0, 1,1, 1,1),
(0,1, 0,0, 0,1),
(0,1, 0,1, 1,0),
(0,1, 1,0, 1,1),
(0,1, 1,1, 0,0),
(1,0, 0,0, 1,0),
(1,0, 0,1, 1,1),
(1,0, 1,0, 0,0),
(1,0, 1,1, 0,1),
(1,1, 0,0, 1,1),
(1,1, 0,1, 0,0),
(1,1, 1,0, 0,1),
(1,1, 1,1, 1,0)}.

In the above code, if each byte is considered to be a binary encoded number between 0 and 3 then the check byte is equal to the sum of the information bytes modulo 4. This is an example of a checksum code [38]. □

Theorem 4.4: A functional block which is a b-byte sliced circuit with an output code space which is a b-byte distance two code where each byte of the code is produced by a single byte slice of the circuit is P-FS with respect to single faults.

Proof: A single fault in a b-byte sliced structure has a sensitization set where all vectors contain 1's or 0's in at most one output byte position. Therefore when any such sensitization vector and output code vertex are operated on by the ⓔ operator, at most one code position can be affected. Since any two code words of a b-byte distance two code must differ in at least two positions, the result of the ⓔ operation must be the correct code word or a noncode word. Q.E.D.

Corollary 4.2: A functional block which is a b-byte sliced with an output code space which is a b-byte distance two code where each byte of the code is produced by a single byte slice of the circuit is E-TSC with respect to single faults.

Proof: From Theorems 2.3 and 4.4.

Corollary 4.3: Any mapping onto a b-byte distance two output code space can be made TSC with respect to single faults.

Proof: Since primary inputs can be shared such a mapping can always be realized with a b-byte sliced circuit. Theorem 4.4 states that this mapping is P-FS, and reducing the circuit with respect to any undetected faults as discussed following Theorem 2.5 yields a TSC circuit.

Q.E.D.

There is a large number of b-byte distance two codes. The checksum code has been found to be a good code for constructing adders [12]. However, the checksum code is difficult to check and to convert to an unordered code. Since the purpose of this thesis is to design circuits for arbitrary functions, we will consider only the distance-two b-adjacent (DTBA) codes [39]. These codes appear to be as easy to use as the checksum codes for arbitrary functions and have the added advantages of being easily checked and converted to unordered codes.

4.4.1. Byte Sliced Circuits with DTBA Codes

We begin by presenting a simple definition of DTBA codes. Consider the bit positions of a code vector x containing n bytes each of length b to be

$$(x_{11},x_{12},\ldots,x_{1b},x_{21},x_{22},\ldots,x_{2b},\ldots,x_{n1},x_{n2},\ldots,x_{nb}).$$

Let the first n-1 bytes be the information positions of the code, and let the last byte be the check byte. The bits of the check byte are determined in the following way:

$$x_{ni} = x_{1i} \oplus x_{2i} \oplus x_{3i} \cdots \oplus x_{n-1i}.$$

That is, x_{ni} is the parity of the bits which are in position i of the other bytes. (We have chosen even parity, but we could have just as easily used odd parity.)

Theorem 4.5: The DTBA codes are b-byte distance two.

Proof: Let any number of bits belonging to any information byte be changed to their complemented values. Then the parities of each of

these positions must also change. Therefore, the check byte would also need to be changed to yield a code word.

Similarly, if any number of check bits is changed, at least one information byte would also need to be changed in order to make the parities correct. Therefore, the code must be at least distance two since at least two bytes must be changed to go from one code word to another.

Q.E.D.

Example 4.4: The DTBA code with 6 information bits and where $b = 3$ is shown below.

```
(0,0,0,  0,0,0,  0,0,0)
(0,0,0,  0,0,1,  0,0,1)
(0,0,0,  0,1,0,  0,1,0)
   :        :        :
(0,1,0,  1,0,1,  1,1,1)
   :        :        :
(0,1,1,  1,1,0,  1,0,1)
   :        :        :
```

etc. □

Theorems 4.4 and 4.5 indicate that if a DTBA code and a byte sliced structure are used, then the resulting functional block must be P-FS with respect to single faults and can be made TSC with respect to single faults by reduction with respect to any undetected faults.

It is obvious that an error produced by one of these blocks due to a single fault is contained in one byte. We will call such errors single byte errors. In an interconnection of these blocks, it would therefore be necessary to make the blocks error preserving with respect to single byte

errors in order for them to be error transmitting. Unfortunately, it is not clear how this could be done while retaining the TSC property of the networks except by translating to an unordered code, performing the function using unordered codes and translating back. Translators will be discussed in the next section. In addition TSC checkers can be constructed for DTBA codes, so interconnections can be made TSC by placing checkers prior to any non-error transmitting blocks.

4.4.2. Translators Between Unordered and DTBA Codes

Since the function to be realized often determines the most efficient coding to be used in a TSC system, it may often be desirable to use a combination of unordered codes and DTBA codes in the same system. This makes it necessary to be able to translate between the two types of codes.

This is actually rather easy since a two-rail code is not only a closed unordered code as we have observed earlier, but it is also a DTBA code with two bytes which uses odd parity. (True values are in one byte; complements are in the other.)

To translate from a DTBA code to an unordered code, we can first translate to a two rail code; then translate the two-rail code to any other unordered code using techniques discussed in the previous chapter.

Given a code vector x with n bytes each of length b where the information is in the first n-1 bytes, we can translate to a two-rail code word by letting the true side of an information bit come directly from the corresponding information bit in the DTBA code, say x_{ij}, and then by forming

the complements as

$$x'_{ij} = \sum_{\substack{k=1 \\ k \neq j}}^{n} x_{ik}.$$

Example 4.5: A translator from the DTBA code with 4 information bits and b = 2 to the two-rail code is shown in Figure 4.8. □

The translators discussed above are bit sliced and must therefore be P-FS with respect to single faults (they are TSC as well).

It is actually not necessary that these translators be bit sliced, however. They are still P-FS if each output pair y,y' is formed by independent logic; that is, different complemented outputs may share logic, and different true outputs may share logic. In this way more economical translators are possible. Nevertheless, we should note that if logic is shared in this way, a fault can result in a non-unidirectional output error, even though the output is not a two-rail word.

We also observe that these translators are error-preserving with respect to single byte errors. That is, if all input errors are restricted to a single byte (which is the error generation capability of a byte sliced structure), then a noncode output word results. Since a two-rail code is unordered and closed,we can make a translator from a two-rail code to any unordered code disjoint according to Corollaries 3.4 and 3.5. Thus, the translator from a DTBA code to any unordered code which consists of a bit sliced TSC translator from the DTBA code to the two-rail code followed by a code disjoint TSC translator from the two-rail code to the unordered code must be TSC and error preserving with respect to single byte errors.

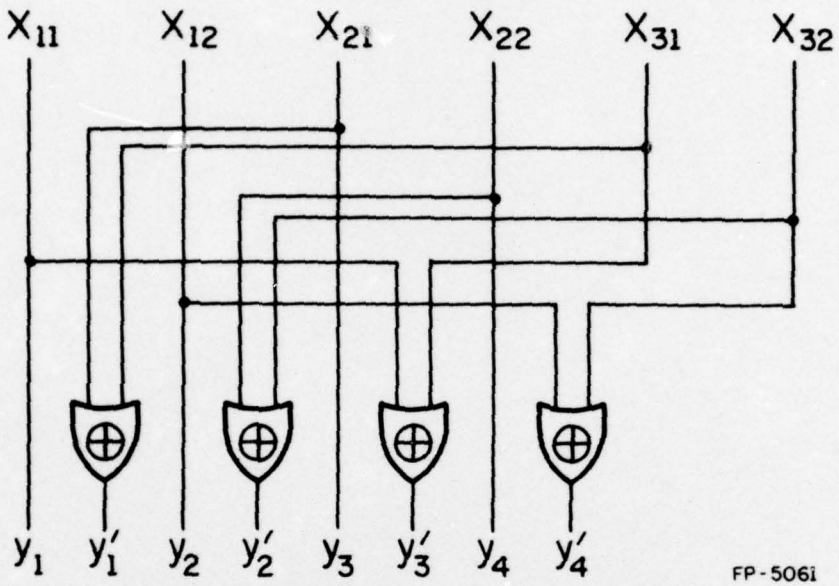


Figure 4.8. A DTBA to two-rail translator.

Example 4.6: A translator from the DTBA code with 3 information bits and b = 1 to the Berger code with 3 information bits is shown in Figure 4.9. □

To translate from an unordered code to a DTBA code, we can first translate the unordered code to a two-rail code, and then translate the two-rail code to the desired DTBA code. Since functional blocks realizing mappings of unordered codes onto unordered codes were discussed in Chapter 3, we will only show how the latter mapping can be realized.

The information bits of the DTBA code can be taken directly from the true bits of the two-rail code, and are thus realized as lines. The bits of the check byte can then be computed from the complemented values of the two-rail code; the realization is not important although an EXCLUSIVE-OR network works quite well.

Such a translator is obviously b-byte sliced, and is in fact TSC if the network realizing the check byte is irredundant. Unfortunately this translator does not necessarily preserve unidirectional errors. For this reason, in order for the unordered code to DTBA translator to be TSC, a TSC checker between the unordered to two-rail translator and the two-rail to DTBA translator may be necessary.

A similar technique to the one just described can be used to translate from one DTBA code to another. In particular, the information bits are translated as lines, and the check bits can be computed from the check bits of the input code and, possibly, from some of its information bits.

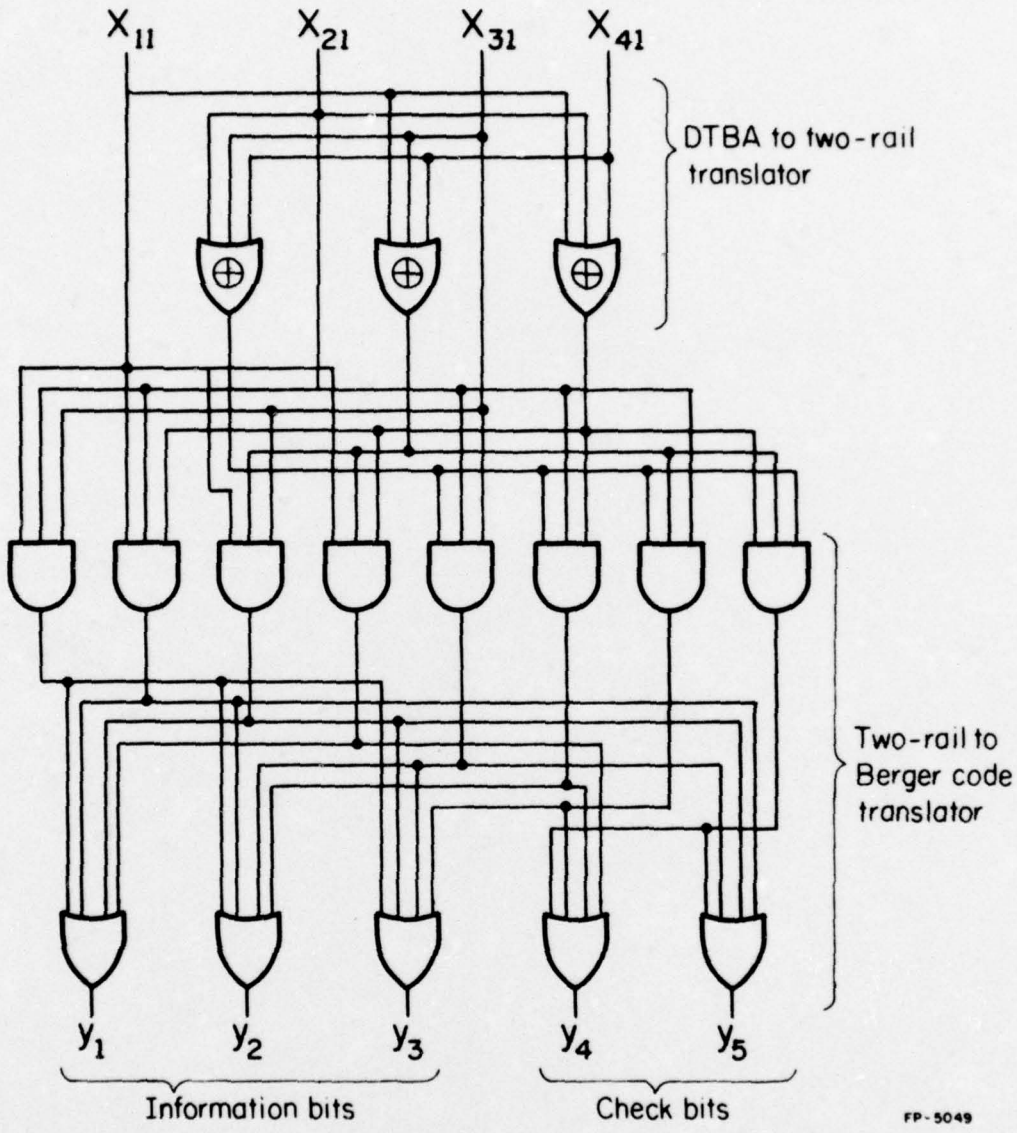


Figure 4.9. A TSC DTBA to Berger code translator.

4.4.3. TSC Checkers for DTBA Codes

A two-output TSC checker for a DTBA code can be constructed by first translating the code to a two-rail code as described in the previous section. Then the two-rail code can be checked by a code disjoint two-rail checker as described in Chapter 3. The checker must be code disjoint, since all errors produced by the first translator are not necessarily unidirectional. These checkers are similar to the DTBA to unordered code translators except that the two-rail to unordered code translator block is replaced with a two-rail TSC checker.

5. BEHAVIOR OF TSC CIRCUITS UNDER NONMODELED FAILURES

5.1. Introduction

In the previous chapters, we have given methods for constructing combinational circuits which are guaranteed to be TSC with respect to some fault set, e.g. unidirectional faults and single faults. However, these circuits are also TSC with respect to other fault sets by virtue of their structure and TSC property for the assumed fault set. We will now establish several fault sets for which the circuits are TSC in addition to being TSC for the explicitly given fault set. We will consider both permanent and nonpermanent faults.

5.2. Behavior of Circuits which are TSC for Unidirectional Faults

We will first consider any faults which change the logic function realized by a functional block in any way. Any unidirectional fault is obviously included, but so are many others. Simply because the output code space is unordered a functional block G is fault secure with respect to $\mathfrak{F}$ where $\mathfrak{F} = \{f | \forall a \in A \;\; G(a,f) \leq G(a,\phi) \text{ or } G(a,\phi) \leq G(a,f)\}$. This fault set $\mathfrak{F}$ is very large. We now present two examples of faults contained in $\mathfrak{F}$.

First we will consider failures which can occur when the lines become shorted together. In many technologies, such a short can be modeled as a wired OR or a wired AND. That is, the OR or AND function is realized at the point where the wires are shorted together. Figure 5.1 illustrates this situation for the case where a wired OR results from a short.

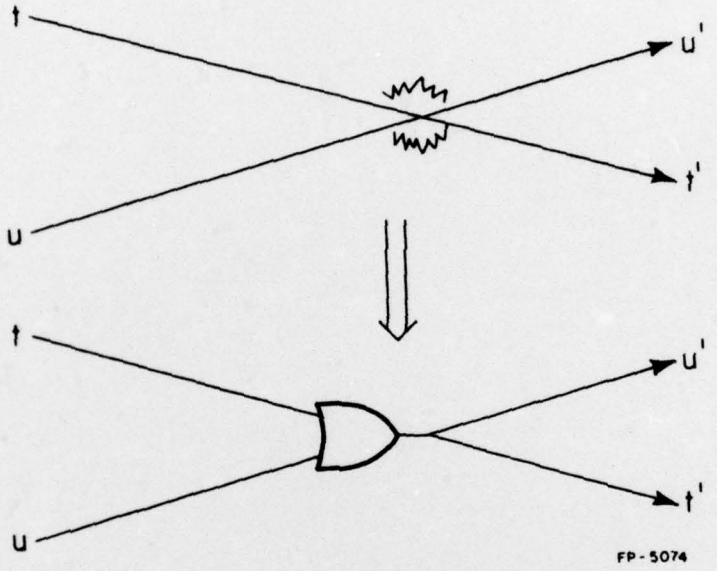


Figure 5.1. A short-circuit between lines resulting in a wired OR.

If a wired OR results from a short as shown in Figure 5.1, then we observe that the pair (t',u') after the short covers the pair (t',u') before the short. If a wired AND results then the pair before the short covers the pair after the short. Due to the inverter-free property of the network, this can only lead to additional 1's at the output in the case of a wired OR and additional 0's at the output in the case of a wired AND. Thus $\mathfrak{F}$ includes all shorts in the inverter-free functional block. This includes shorts which create feedback, since feedback in an inverter-free situation cannot cause the value on the line feeding back to change.

If a short occurs we are only assured that an inverter-free functional block is fault secure with respect to the short. However, if the short does not affect the output, the circuit will still be inverter-free and will thus be E-TSC with respect to unidirectional faults and fault secure with respect to any additional shorts.

If a line should become stuck at a level between a logical 1 and a logical 0 (stuck-at-$\frac{1}{2}$), then some gates being fed by the line which is stuck-at-$\frac{1}{2}$ may interpret it as a 1, and some may interpret it as a 0, depending perhaps on the input. However, one of these two values is correct, and the other can be modeled as a unidirectional stuck-at fault. Therefore $\mathfrak{F}$ includes all single stuck-at-$\frac{1}{2}$ faults. If some stuck-at-$\frac{1}{2}$ fault does not affect the output, a second stuck-at-$\frac{1}{2}$ fault or unidirectional fault could conceivably conspire with it to yield an erroneous code output. Thus we can only say that an inverter-free functional block is fault secure with respect to single stuck-at-$\frac{1}{2}$ faults.

5.3. Behavior of Circuits which are TSC for Single Faults

Consider a network G which is TSC with respect to single faults containing a single-output subnetwork N. Define T to be a set of tests from the input code space of G such that each member of T detects some single fault in N. Let T' be the set of vertices which appear at the inputs of N due to a member of T being present at the inputs of G. Then if a fault f changes the function realized by N in any way (not just stuck lines) and if $\exists t \in T'$ $N(t,f) \neq N(t,\phi)$ then G is TSC with respect to f.

For example, let the network N be a 4-input NAND gate with inputs i_1, i_2, i_3, i_4. If a short between inputs i_1 and i_2 results in a wired OR, then the function realized by N becomes $\overline{(i_1 \vee i_2) i_3 i_4}$. Since G is TSC with respect to single faults, the vertices (0,1,1,1),(1,0,1,1),(1,1,0,1), (1,1,1,0),(1,1,1,1) all must arrive at the inputs of N during normal operation with the output of N sensitized to some output of G. The output of N before the short under (0,1,1,1) is 1, and the output of N after the short under the same input is 0. Therefore, the circuit G is TSC with respect to this short-circuit fault.

By generalizing the above example, it can be shown that if the shorting of the inputs of some gate yields a wired OR or AND then a circuit which is TSC with respect to single faults is TSC with respect to gate input shorts which are internal to the gate.

Consider a circuit G which is P-FS with respect to single faults. When any single output subnetwork of G fails so that it performs a different function one of two things happens: 1) the new function coincides with the original one for every input which affects the output or

2) the new function and the original one differ for at least one input so that the output is affected. If 1) happens, then the resulting circuit continues to operate properly, and is still P-FS by Theorem 4.2, so it is still E-TSC. If 2) happens, the output which is affected must be a noncode output due to the P-FS property of the circuit. Thus, a circuit which is P-FS with respect to single faults is also P-FS with respect to any single-output subnetwork changing its function. This obviously includes shorts within a single-output subnetwork such that feedback is not formed, (feedback can yield oscillations among other problems), as well as single stuck-at-$\frac{1}{2}$ faults.

5.4. Nonpermanent Faults

We consider two types of nonpermanent faults--transients and intermittents. A transient fault appears only once for some (usually short) period of time and never reappears. An intermittent appears, disappears, and reappears many times.

If a circuit is TSC with respect to some logical fault set $\mathfrak{F}$, and some member of $\mathfrak{F}$ appears only as a transient, then disappears, it is obvious that if the transient causes the output to be incorrect, it must cause it to be noncode output. So, we see that if a circuit is TSC with respect to $\mathfrak{F}$, it is also protected from members of $\mathfrak{F}$ which appear as transients.

If a member of $\mathfrak{F}$ appears an an intermittent, we would like to detect its presence before a second fault can occur. Assume the fault

appears randomly and is present 1/t of the time. By considering the MTBF of the circuit for permanent failures, let T be a period of time within which all code inputs must be applied to the circuit to yield reliable operation, i.e. to insure within a reasonable probability that any member of $\mathfrak{F}$ is detected before a second member of $\mathfrak{F}$ can occur. Then, to insure reliable operation under the presence of the intermittent failure it is necessary to cycle through all code inputs every T/t seconds. The derivation of this ratio should be obvious.

For a nonrandom intermittent fault, the problem becomes much more complex since some nonrandom input sequence may be required in order to detect it.

6. SUMMARY AND CONCLUSIONS

6.1. Summary of Thesis

The purpose of this thesis has been to present general methods for constructing totally self-checking combinational logic circuits. The methods which are given can be used for constructing circuits which perform arbitrary functions--not just special functions such as binary addition.

The thesis begins by considering the class of hardware failures which can be modeled as permanent stuck-at faults. Using this rather general fault model, the interconnection of TSC functional blocks is discussed.

After discussing the path-fault secure property, the fault model is reduced to unidirectional stuck-at faults and single stuck-at faults with a chapter devoted to the construction of circuits which are TSC with respect to each of the two fault sets.

The circuits which are designed to be TSC with respect to permanent stuck-at faults are also TSC with respect to faults which are nonpermanent and/or nonstuck. The final chapter discusses these nonmodeled faults for which a circuit may be TSC.

Three new concepts are introduced in this thesis. They are the properties of being effectively-totally self-checking, path-fault secure, and error transmitting.

Allowing effectively-totally self-checking circuits makes the interconnection of small TSC functional blocks easier since some interconnections which are E-TSC may not be TSC. However, a E-TSC circuit gives fault protection which is just as good as that which a TSC circuit gives.

Using path-fault secure circuit structures greatly simplifies TSC design. It effectively splits the design procedure into two parts. The first part consists of selecting a circuit structure and output encoding which give the path-fault secure property. Then, the circuit can be made self-testing by simply removing lines without disturbing the fault security of the circuit.

The use of circuits which are error transmitting allows more relaxed constraints on the mapping of their noncode space inputs than those which are code disjoint. Using code disjoint circuits requires that all noncode input words map to noncode outputs. However, a circuit which is error transmitting only needs to map noncode inputs which can potentially occur under the fault assumption to noncode outputs.

6.2. Suggested Future Research

Many of the rules for designing TSC circuits which are given in this thesis simply say that if the desired mapping satisfies certain conditions, then any circuit realization satisfying certain constraints must be TSC. This allows a variety of circuit structures to be used. Some of the most promising are array structures. This is because the use of TSC circuits is most easily justified in LSI technologies, and LSI technologies demand regular circuit structures. Thus, it appears that TSC logic arrays should be studied. Furthermore, it appears that the regularity of some of the codes we have discussed, e.g. m-out-of-n codes, makes the design of TSC logic

arrays a natural thing to do. For example, the TSC checker for k-out-of-2k codes given in Chapter 4 is the best known to date on a number of counts, and is also the most regular.

New codes and circuit structures should also be examined. The path-fault secure property should be helpful in this type of research, since it simplifies the problem. It might also be possible to find other properties like the path-fault secure property which guarantee fault secureness and are invariant under the removal of untested lines.

It may also be possible that some of the results given here can be extended to circuits with feedback. This could lead to a better understanding of TSC sequential machines.

REFERENCES

1. Tryon, J. G., "Quadded Logic," in Redundancy Techniques for Computing Systems, Wilcox and Mann, eds., pp. 205-228, Spartan Books, Washington, D. C., 1962.

2. Von Neumann, J., "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," Annals of Mathematical Studies, No. 34, pp. 43-98, Princeton University Press, Princeton, N.J., 1956.

3. Avizienis, A., "Design of Fault Tolerant Computers," Proc. of the Fall Joint Computer Conference, pp. 733-743, 1967.

4. Mathur, F. P., "Reliability Modeling and Analysis of a Dynamic TMR System Utilizing Standby Spares," Proc. of Seventh Annual Allerton Conf. on Circuit and System Theory, Monticello, Illinois, pp. 243-252, Oct. 1969.

5. Roth, J. P., et al., "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," IEEE Trans. on Computers, Vol. C-16, pp. 567-579, Oct. 1967.

6. Carter, W. C. and P. R. Schneider, "Design of Dynamically Checked Computers," IFIP 68, Vol. 2, Edinburg, Scotland, pp. 878-883, Aug. 1968

7. Anderson, D. A., "Design of Self-Checking Digital Networks Using Coding Techniques," Coordinated Science Laboratory Report R-527, University of Illinois, Sept. 1971.

8. Sellers, F. F., H. Y. Hsiao, and L. W. Bearnson, Error Detecting Logic for Digital Computers, McGraw-Hill, New York, 1968.

9. Elias, P., "Computation in the Presence of Noise," IBM Journal, Vol. 2, pp. 346-353, Oct. 1958.

10. Massey, J. L., "Survey of Residue Coding for Arithmetic Errors," ICC Bulletin, Vol. 3, pp. 241-248, Rome Italy, Oct. 1964.

11. Avizienis, A., "Digital Fault Diagnosis by Low-Cost Arithmetic Coding Techniques," Tech. Report No. 32-1476, Jet Propulsion Laboratory, Aug. 1970.

12. Wakerly, J. F., "Checked Binary Addition Using Parity Prediction and Checksum Codes," Tech. Note No. 39, Digital Systems Laboratory, Stanford University, Jan. 1974.

13. Ho, D., "The Study of a Totally Self-Checking Adder," Coordinated Science Laboratory Report No. R-582, University of Illinois, Aug. 1972.

14. Smith, J. E. and G. Metze, "General Design Rules for the Construction of m-out-of-n Totally Self-Checking Checkers," Proc. of 13th Annual Allerton Conf. on Circuit and System Theory, Monticello, Illinois, pp. 704-715, Oct. 1975.

15. Reddy, S. M., "A Note on Self-Checking Checkers," IEEE Trans. on Computers, Vol. C-23, pp. 1100-1102, Oct. 1974.

16. Bouricius, W. G., et al., "Interactive Design of Self-Testing Circuitry," Purdue Centennial Symposium on Information Processing, pp. 73-80, April 1969.

17. Huffman, D. A., "Combinational Circuits with Feedback," in Recent Developments in Switching Theory, A. Mukhopadhyay, ed., pp. 28-55, Academic Press, New York, 1971.

18. Chang, H. Y., E. G. Manning, and G. Metze, Fault Diagnosis of Digital Systems, Wiley-Interscience, New York, 1970.

19. Pierce, W. H., Failure Tolerant Computer Design, Academic Press, New York, 1965.

20. Diaz, M., "Design of Totally Self-Checking and Fail-Safe Sequential Machines," Digest of the Fourth Annual Symp. on Fault-Tolerant Computing, pp. 3-19 to 3-24, June 1974.

21. Mine, H. and Y. Koga, "Basic Properties and a Construction Method for Fail-Safe Logical Systems," IEEE Trans. on Electron. Computers, Vol. EC-16, pp. 282-289, June 1967.

22. Preparata, F. P. and R. T. Yeh, Introduction to Discrete Structures, Addison-Wesley, Reading, Mass., 1974.

23. Kohavi, Z., Switching and Finite Automata Theory, McGraw-Hill, New York, 1970.

24. Betancourt, R., "Derivation of Minimum Test Sets for Unate Logical Circuits," IEEE Trans. on Computers, Vol. C-20, pp. 1264-1269, Nov. 1971.

25. Ramsey, F. P., "On a Problem of Formal Logic," Proc. London Math. Soc., 2nd Ser., Vol. 30, pp. 264-286, 1930.

26. Liu, C. L., Topics in Combinatorial Mathematics, Notes on lectures given at the 1972 MAA Summer Seminar, Williams College, Williamstown, Mass., Math. Assoc. of America, 1972.

27. Kalbfleisch, J. G., "On the Ramsey Number N(4,4;3)," in Recent Progress in Combinatorics, W. J. Tutte, ed., Academic Press, New York, pp. 273-282, 1969.

28. Isbell, J. R., "N(4,4;3) $\geq$ 13," Journal of Comb. Theory, Vol. 6, p. 210, 1969.

29. Lubell, D., "A Short Proof of Sperner's Lemma," Journal of Comb. Theory, Vol. 1, p. 299, Sept. 1966.

30. Schonheim, J., "On Coverings," Pacific Journal of Math., Vol. 14, pp. 1405-1411, 1964.

31. Kalbfleisch, J. G. and R. G. Stanton, "Maximal and Minimal Coverings of (k-1)-tuples by k-tuples," Pacific Journal of Math., Vol. 26, No. 1, pp. 131-140, 1968.

32. Anderson, D. A. and G. Metze, "Design of Totally Self-Checking Check Circuits for m-out-of-n Codes," IEEE Trans. on Computers, Vol. C-22, pp. 263-269, March 1973.

33. Knuth, D. E., The Art of Computer Programming, Vol. 1, Addison-Wesley, Reading, Mass., 1968.

34. Dilworth, R. P., "A Decomposition Theorem for Partially Ordered Sets," Ann. of Math., Vol. 51, pp. 161-166, 1950.

35. Berger, J. M., "A Note on Error Detection Codes for Asymmetric Channels," Information and Control, Vol. 4, pp. 68-73, March 1961.

36. Ashjaee, M. J. and S. M. Reddy, "Totally Self-Checking Checkers for a Class of Separable Codes," Proc. of 12th Annual Allerton Conf. on Circuit and System Theory, Monticello, Illinois, pp. 238-243, Oct. 1974.

37. Hayes, J. P., "A NAND Model for Fault Diagnosis in Combinational Logic Networks," IEEE Trans. on Computers, pp. 1495-1506, Dec. 1971.

38. Garner, H. L., "Generalized Parity Checking," IRE Trans. on Electron. Computers, Vol. EC-7, pp. 207-213, 1958.

39. Bossen, D. C., "B-Adjacent Error Correction," IBM Journal of Research and Development, Vol. 14, pp. 402-408, 1970.

40. Armstrong, D. B., "On Finding a Nearly Minimal Set of Fault Detection Tests for Combinational Logic Networks," IEEE Trans. on Electron. Computers, Vol. EC-15, pp. 66-73, Feb. 1966.

41. Ozguner, F., "Design of Totally Self-Checking Asynchronous Machines," Coordinated Science Laboratory Report No. R-679, University of Illinois, May 1975.

42. Schertz, D. R. and G. Metze, "A New Representation for Faults in Combinational Digital Circuits," IEEE Trans. on Computers, pp. 1361-1364, Nov. 1971.

VITA

James Edward Smith was born in Quincy, Illinois on November 29, 1950. He received a B.S. degree in electrical engineering/computer science in 1972 and an M.S. degree in computer science in 1974 both from the University of Illinois. He was a research assistant with the Digital Systems Group at the Coordinated Science Laboratory from 1973-1976.